Maisterintutkielma

Tietojenkäsittelytiede

Jatkuva integraatio ja toimittaminen reguloidussa ohjelmistopohjaisten lääkinnällisten laitteiden kehityksessä

Ilari Richardt

8.3.2020

Matemaattis-luonnontieteellinen tiedekunta

Helsingin yliopisto

Ohjaaja(t)

Prof. Tommi Mikkonen

Tarkastaja(t)

Niko Mäkitalo

Yhteystiedot

PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

Sähkopostiosoite: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytiede

Tekijä — Författare — Author

Ilari Richardt

Työn nimi — Arbetets titel — Title

Jatkuva integraatio ja toimittaminen reguloidussa ohjelmistopohjaisten lääkinnällisten laitteiden kehityksessä

Ohjaajat — Handledare — Supervisors

Prof. Tommi Mikkonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Maisterintutkielma	8.3.2020	65 sivua

Tiivistelmä — Referat — Abstract

Ohjelmistopohjaisten lääkinnällisten laitteiden kehitys on murroksessa. EU:n uusi Medical Device Regulation -lainsäädäntö on tullut voimaan ja sen myötä regulaatio lääkinnällisille laitteille muuttuu. Muuttuneiden vaatimusten myötä kehitys ei ohjelmistojen osalta helpotu, vaan päinvastoin. Näin ollen tutkimmekin, miten lääkinnällisten laitteiden ohjelmistokehityksessä voidaan automaation avulla saavuttaa jatkuva integraatio ja tuotantoonvienti, ja näin ollen ketteröittää ohjelmistokehitystä säilyttäen kilpailukyvyn kasvavilla terveydenhuollonalan markkinoilla.

Tutkimme eri luokitusten ja MDR lainsäädännön vaikutusta ohjelmistokehitykseen. Vertailimme erityisesti luokan 2a ohjelmistopohjaisten lääkinnällisten laitteiden kehitystä suhteessa reguloimattomaan ohjelmistokehitykseen tutkimalla standardeja ja osoittamalla niissä olevien vaatimusten automaatiomahdollisuuksia.

Ketteryyden taso kuvaa kyvykkyyttä reagoida muutoksiin ja tuottamaan uusia inkrementtejä ohjelmistokehitysprosessissa. Tutkielman lopputuloksena olemme pystyneet osoittamaan, että lähes sama ketteryyden taso on saavutettavissa ohjelmistopohjaisten luokan 2a lääkinnällisten laitteiden kehityksessä kuin reguloimattomassa ohjelmistokehityksessä, kun huomioon otetaan itse ohjelmistokehitysprosessi, jota erityisesti standardi IEC 62304 kuvaa. Katseltaessa kokonaisprosessia, joka sisältää muitakin osia kuin ohjelmistokehityksen, ei samaa ketteryyden tasoa saavuteta kuin reguloimattomassa ohjelmistokehityksessä. Tämä johtuu regulatiivisten manuaalisten tehtävien luonteesta, jotka on suoritettava ennen tuotantoonvientiä.

ACM Computing Classification System (CCS)
Software and its engineering → Software creation and management → Software development process management → Software development methods → Agile software development

Avainsanat — Nyckelord — Keywords

ohjelmistopohjainen lääkinnällinen laite, jatkuva tuotantoonvienti, jatkuva integraatio

Säilytyspaikka — Förvaringsställe — Where deposited

Helsingin yliopiston kirjasto

Muita tietoja — övriga uppgifter — Additional information

Ohjelmistojärjestelmien erikoistumislinja

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Computer Science

Tekijä — Författare — Author

Ilari Richardt

Työn nimi — Arbetets titel — Title

Jatkuva integraatio ja toimittaminen reguloidussa ohjelmistopohjaisten lääkinnällisten laitteiden kehityksessä

Ohjaajat — Handledare — Supervisors

Prof. Tommi Mikkonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	March 8, 2020	65 pages

Tiivistelmä — Referat — Abstract

Medical Device software, especially software based medical device, development is currently at a turning point. EU's new Medical Device Regulation (MDR) has entered into force and its changing the way medical devices are developed. Changed requirements don't ease software development on the contrary it's making it harder at some points. This study is about automating software based medical device's development process, and demonstrating how it can be achieved to maintain competitiveness in growing healthcare market.

We studied different classifications and MDR legislation's impact on software development, comparing especially class 2a medical devices with non-regulated software development. We identified requirements from standards and demonstrated automation possibilities to tackle them.

With this study we have been able to demonstrate that almost the same level of agility and speed of software increments can be achieved in class 2a medical device software development when we're taking into account the software development process part of the whole process (described in IEC 62304). When looking at the whole process there are parts (for example usability testing) that cannot be automated, therefore these parts of whole process cannot achieve same level of agility in medical devices software development as in non-regulated software development due to manual nature of these parts of the process.

ACM Computing Classification System (CCS)
Software and its engineering → Software creation and management → Software development process management → Software development methods → Agile software development

Avainsanat — Nyckelord — Keywords

ohjelmistopohjainen lääkinnällinen laite, jatkuva tuotantoonvienti, jatkuva integraatio

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software Systems specialisation line

Sisältö

Sanasto ja määritelmät

CE-merkintä osoittaa, että mm. lääkinnällinen laite on markkinoille kelpaava. Lääkinnällisten ohjelmistopohjaisten laitteiden kontekstissa tämä käytännössä tarkoittaa, että laite täyttää lääkinnällisille laitteille kansainvälisesti hyväksytyt standardit ja laite on kehitetty niiden mukaisesti (mm. ISO 13485 ja IEC 62304).

DevOps on valikoima tietotekniikan prosessin kyvykkyyksiä, jolla pyritään yhdistämään kehitys ja tuotanto mahdollisimman lähelle toisiaan (Smeds et al., 2015). Termi muodostuu lyhenteistä Dev (engl. Development) ja Ops (engl. Operations).

Integraatiotesti tarkoittaa testiä, jolla testataan ohjelmistokomponenttien toimivuus yhdessä, eli sillä testaan ohjelmistokomponenttien integraatio.

Jatkuva integraatio / CI (engl. Continous Integration) on toimintatapa, jossa ohjelmistokehitystiimin jäsenet tuottavat muutoksia versiohallintaan. Jokainen muutos verifioidaan automaattisesti, jotta voidaan varmistua, että muutokset eivät rikkoneet mitään olemassa olevaa mm. testiautomaation avulla. Tämä mahdollistaa nopean kehityksen syklin ketterässä ohjelmistokehityksessä (Meyer, 2014).

Jatkuva tuontatoonvienti / CD (engl. Continous Deployment) on toimintatapa, jossa ohjelmistokehitystiimi tuottaa jatkuvalla syklillä tuotantoon julkaistavaa ohjelmistoa, joka myös viedään tuotantoon jatkuvalla syklillä. Tämän toimintatavan edellytyksenä on vahva automaatio niin CI:n, kuin tuotannon ympäristöönviennin osalta.

Jäljitettävyys tarkoittaa tässä yhteydessä kyvykkyyttä tunnistaa ja yhdistää ohjelmistoon kohdistuvat muutokset kehitysprosessin eri vaiheissa (alkaen suunnittelusta ja päätyen julkaistavaan versioon) (ISO 13485, 2016).

Kehityssykli tarkoittaa tässä tutkielmassa ohjelmistokehityksen yhtä iteraatiota, eli kehitysprosessin vaiheita määrittelystä tuotantoonvientiin, joka toistuu uudestaan ja uudestaan. Yksi DevOps:in päätavoitteista on saada kehityssyklin aikajänne mahdollisimman pieneksi (Smeds et al., 2015).

Lean on ajatusmalli, jossa pyritään luomaan arvoa loppuasiakkaalle, poistaen kaikki turha arvoa tuottamaton osa esimerkiksi kehitysprosessista (Abrahamsson et al., 2012).

Liitospyyntö (engl. Pull Request) on käsite, jolla kuvataan versiohallinnan haaran

liitospyyntöä toiseen haaraan, jonka lopputuloksena kaksi ohjelmiston versiota yhdistetään.

Lääkinnällinen laite on mikä tahansa instrumentti, laitteisto, väine, koje, laite, implantti, in vitro -reagenssi, *ohjelmisto*, materiaali tai muu samankaltainen tai liittyvä tarvike, jolla on tarkoitus tehdä mm. sairauden diagnosointia, ehkäisyä, tarkkailua, hoitoa tai lievitystä (ISO 13485, 2016).

MDR (engl. Medical Device Regulation) on Euroopan Unionin lääkinnällisille laitteille 5.4.2017 voimaan tullut regulaatio, joka ottaa kantaa miten lääkinnällisiä laitteita tulee kehittää (Medical Device Regulation, European Union, 2017).

Ohjelmisto on integroitu kokonaisuus *ohjelmistokappaleita*, jotka toimivat yhdessä toteuttaen ohjelmiston (IEC 62304, 2015). Tätä integraatiota voidaan automatisoida CI metodeilla (Meyer, 2014).

Ohjelmistokappale (engl. Software Item) on tunnistettava osa ohjelmaa, kuten lähdekoodi tai ohjelmiston data (IEC 62304, 2015).

Ohjelmistopohjainen lääkinnällinen laite on lääkinnällinen laite, mikä koostuu puhtaasti vain ohjelmisto-komponentista. Tällaisia lääkinnällisiä laitteita ovat mm. tekoäly-pohjaiset hoidon tarpeen arvioinnit. Näillä lääkinnällisillä laitteilla ei siis ole fyysistä muotoa, kuten stetoskoopilla tai sydänkeuhkokoneella. Kuitenkin esimerkiksi potilastietojärjestelmät, PACS-järjestelmät (radiologisten kuvien arkistointijärjestelmät) tai muut vastaavat arkistointia toteuttavat, papereita korvaavat järjestelmät eivät ole lääkinnällisiä laitteita (Medical Device Regulation, European Union, 2017).

Ohjelmistoyksikkö (engl. Software Unit) on osa ohjelmistoa, jota ei jaeta enää pienemmäksi (IEC 62304, 2015).

Riski on vahingon ilmenemistodennäköisyys yhdistettynä vahingon vakavuuteen (ISO 13485, 2016)

SOUP (engl. Software of Unknown Provenance) on jo valmiiksi kehitetty ohjelmistokappale, kuten avoimen lähdekoodin kirjasto, jota ei ole suunniteltu lääkinnällisen laitteen vaatimuksia täyttäväksi tai sen kehitysprosessi on tuntematon (IEC 62304, 2015). Etenkin moderneja JavaScript-pohjaisia ohjelmistoja kehitettäessä tulee helposti käytettyä satoja tai tuhansia riippuvuuksia yksittäisten kirjastojen kautta (Decan et al., 2017).

Staattinen analyysi tarkoittaa ohjelmiston katselmointia (joko ihmisen tai automatiikan toimesta) ilman, että sitä suoritetaan.

Sulautettu ohjelmisto (engl. Embedded Software) on ohjelmisto, joka toimii sulautetusti yhdessä laitteiston kanssa.

Järjestelmätestaus (engl. System Testing) tarkoittaa koko järjestelmän kokonaistestausta, että se täyttää sille asetetut vaatimukset kokonaisuudessaan

Valmistaja on se luonnollinen- tai oikeushenkilö, joka asettaa lääkinnällisen laitteen markkinoille riippumatta siitä, onko se itse toteuttanut sitä (ISO 13485, 2016).

Verifiointi on varmistus, että vaatimukset esim. muutokselle täyttyvät (IEC 62304, 2015)

V-malli (myös Validation & Verification model) on ohjelmiston kehitysmalli, jossa edetään lineaarisesti aloittaen ylätason määrittelyistä ja tarkennetaan määrittelyitä alemmille tasoille. Tämän perusteella tehdään toteutus, jonka jälkeen jokainen määrittelyn taso verifioidaan sitä vastaavalla testisetillä (Balaji ja Murugaiyan, 2012).

Yksikkötesti on yhden ohjelmistoyksikön yksittäinen testi, jolla testataan ohjelmistoyksikön toimivuutta tietyillä lähtöarvoilla.

1 Johdanto

Terveydenhuollon ala digitalisoituu jatkuvasti enemmän ja sen digitalisaatioon investoidaan enemmän (Day ja Zweig, 2018). Merkittävä osa investoinneista menee ohjelmistojen kehitykseen (*Healthcare Software Investment Trends - Digital Health Matters* 2017). Terveydenhuollon alalla osa ohjelmistosta lasketaan lääkinnällisiksi laitteiksi niiden käyttötarkoituksen myötä (Medical Device Directive, European Union, 1993). Suurin osa standardeista, jotka implementoivat lainsäädäntöä, perustuvat vesiputouskehitysmalliin, jossa kehitysaskeleet ovat irrallaan toisistaan ja vaiheesta toiseen siirtyminen edellyttää edellisen vaiheen valmistumista – eteneminen tapahtuu siis lineaarisesti. Nyt kehityksen lisääntyessä ja nopeutuessa tarvitsemme tehokkaampia tapoja lääkinnällisten laitteiden ohjelmistokehitykseen.

Perinteisesti lääkinnällisiksi laitteiksi on mielletty fyysisessä muodossa olevat laitteet, kuten sydän-keuhkokoneet, sairaalavuoteet tai kävelykepit. Osassa lääkinnällisistä laitteista on myös ollut ohjelmistoa, jolla laitteen toimintaa ohjataan. Esimerkiksi sydän-keuhkokoneessa olevalla ohjelmistolla on merkittävä vaikutus laitteen toimintaan. Nykyään yhä useampi lääkinnällinen laite on ohjelmisto, johon ei liity mitään fyysistä olomuotoa, kuten mm. laboratoriossa käytettävät ohjelmistot, lääkäreiden päätöksenteon tukijärjestelmät (engl. DSS, Decision Support System), sekä jatkossa yhä enemmän yleistyvät AI-/tekoälysovellukset.

Lääkinnällisille laitteille on määritetty laatuvaatimuksia niiden eri luokkien mukaisesti, joita ovat 1, 2a, 2b ja 3. Valmistaja luokittelee, mihin luokkaan lääkinnällinen laite kuuluu. Luokittelun taustalla on ajatus siitä, mitä potilaalle voi tapahtua laitteeseen liittyvän riskin realisoituessa. Alimman tason luokka 1 sisältää laitteita, jotka eivät aiheuta välitöntä potilasturvallisuusriskiä. Tässä luokassa olevat laitteet ovat tyypillisesti esimerkiksi kuumemittareita, silmälaseja tai kävelykeppejä. Luokka 2a, johon tässä tutkielmassa erityisesti keskitytään, sisältää erityisesti ohjelmistopuolella sellaisia ohjelmistoja, joiden perusteella lääketieteen ammattihenkilöstö tekee diagnostisia päätöksiä tai niitä käytetään terapeuttisiin käyttötarkoituksiin. Tällaisia laitteita ovat mm. laboratoriossa käytettävät ohjelmistot tai päätöksenteon tukijärjestelmät, mikäli ne eivät voi aiheuttaa peruuttamatonta haittaa tai kuolemaa potilaalle. Luokan 2a laitteet voivat aiheuttaa potilaalle haittaa, mutta ne rajoittuvat vain välilliseen hait-

taan. Luokkaan 2b kuuluvat laitteet, joiden virheet voivat johtaa potilaan tilapäiseen haittaan. Näitä laitteita ovat mm. automaattiset lääkkeiden säännöstelijät ja potilasmonitorit. Korkeimman luokan 3 laitteisiin kuuluu laitteet, joiden virhetilanteet voivat aiheuttaa potilaan välittömän kuoleman tai vakavan peruuttamattoman vamman, näitä laitteita ovat mm. sydän-keuhkokoneet.

Vuonna 2020 Euroopan Unionin MDR:n (Medical Device Regulation) siirtymäaika päättyy, jonka myötä lääkinnällisten laitteiden luokittelusäännöt tiukentuvat ohjelmistojen osalta. Ohjelmistot, jotka ovat MDR:ään edeltäneellä MDD (engl. Medical Device Directive) -lainsäädännön aikakaudella kuuluneet luokkaan 1, kuuluvat uuden luokittelun mukaan käytännössä aina luokkaan 2a, joka tuo mukanaan uusia laatuvaatimuksia sovellusta kehittäville tiimeille.

Osana modernia sovelluskehitystä käytetään nykyään paljon jatkuvaa integraatiota (CI, engl. Continious Integration) ja jatkuvaa tuotantoonvientiä (CD, engl. Continious Deployment), jolla sovelluksen integriteetti varmistetaan automaattisesti sekä automatisoidaan tuotantoon vienti niin, että sovellus voidaan viedä tuotantoon tiheään tahtiin ja milloin vain. Luokan 2a lääkinnällisten laitteiden kehitysvaatimukset kuitenkin kuvaavat pitkälti vesiputousmallista kehitystä, jossa laadunvarmistus tapahtuu suunnittelusta tuotantoon vientiin asti ja jokaista tuotantoon vietävää versiota vasten pitää rakentaa dokumentaatio, sekä suorittaa laatuvalidointi mm. testauksen avulla.

Tässä tutkielmassa keskitytään puhtaasti ohjelmistoina esiintyviin lääkinnällisiin laitteisiin – ohjelmistopohjaisiin lääkinnällisiin laitteisiin. Käymme läpi, miten ohjelmiston, jonka käyttötarkoitus täyttää lääkinnällisen laitteen 2a tason määrityksen, jatkuva integraatio ja tuotanto vienti voidaan saavuttaa ja täyttää samalla standardin mukaiset laatuvaatimukset, jotta esimerkiksi CE-merkintä lääkinnälliselle laitteelle olisi mahdollista hakea.

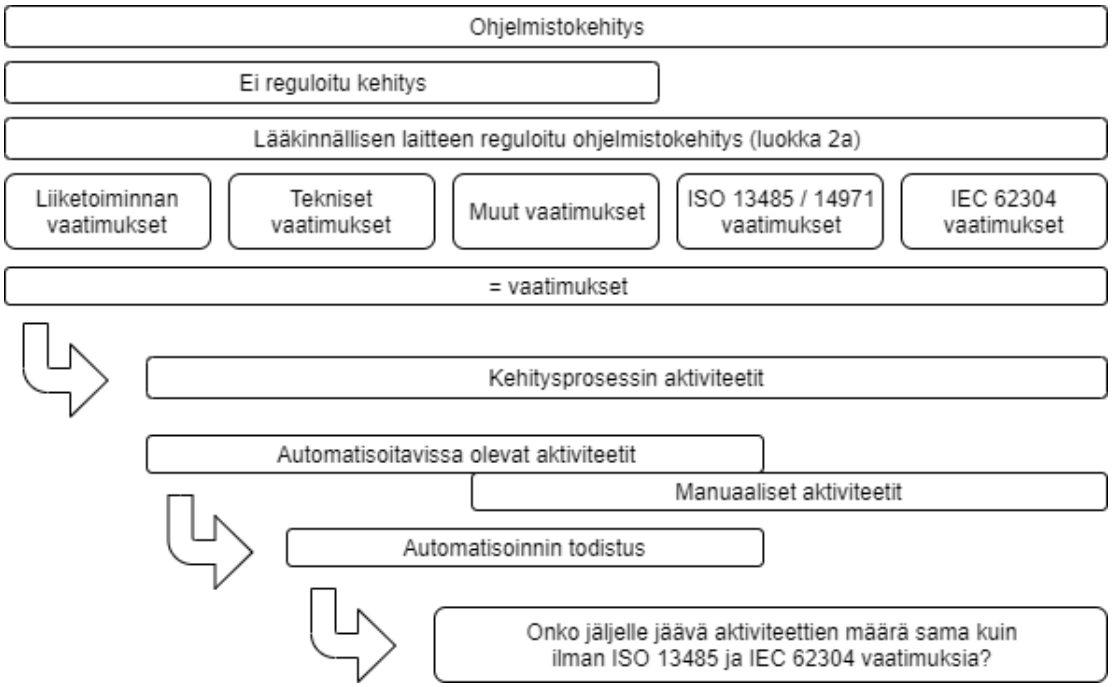
Työn tutkimuskysymykset ovat:

RQ1: Mitkä ovat luokkien 1 ja 2a lääkinnällisten laitteiden erot?

RQ2: Miksi ohjelmistopohjaiset lääkinnälliset laitteet ovat EU:n uuden lainsäädännön mukaisesti lähtökohtaisesti luokassa 2a eivätkä luokassa 1?

RQ3: Voidaanko luokan 2a ohjelmistopohjaisen lääkinnällisen laitteen jatkuva integraatio ja tuotantoonvienti automatisoida ja miten?

Tutkimushypoteesina oli, että reguloidussa ohjelmistokehityksessä pystytään saavuttamaan sama tuotantoonvientinopeus toteutusvaiheesta tuotantovalmiuteen. Tähän ei



Kuva 1.1: Tutkimuksen rakenne

siis oteta mukaan mm. suunnitteluun, käytettävyystestaukseen ja validointiin kuluvaa aikaa, jotka ovat lääkinnällisille laitteille äärimmäisen tärkeitä. Kriittisinä keinoina tämän saavuttamisessa ovat jatkuva integraatio sekä tuotantoonvienti ja merkittävän pitkälle viety automaatio. Tutkielmassa käydään myös läpi, millaisia automaatio-operaatioita jatkuvan integraation ja tuotantoonviennin tulee pystyä tekemään, sekä miten niiden muutoshallinta pitää järjestää, jotta niiden läpi menevä sovellus pystyy täyttämään luokan 2a vaatimustason.

Kuvassa 1.1 on avattu rakennetta, jolla tutkielman **RQ3**-tutkimuskysymystä on lähdetty tutkimaan. Pyrimme siis tunnistamaan kaikki ohjelmistokehityksen vaatimukset ja erottelemaan ne reguloituun ja ei-reguloituun ohjelmistokehitykseen, jonka jälkeen pyrimme automatisoimaan reguloituun ohjelmistokehitykseen liittyvät aktiviteetit. Automaation todistuksen jälkeen voimme arvioida onko jäljelle jäävien aktiviteettien manuaalikuormitus yhtä iso reguloidussa ja reguloimattomassa ohjelmistokehityksessä.

Tutkielman teknisenä kontribuutiona toteutettiin simulaatiomaisesti luokan 2a ohjelmistopohjaisen lääkinnällisen laitteen jatkuvan integraation ja tuotantoonviennin malli, jonka avulla erityisesti automaation mahdollisuuksia todistettiin.

Tuotannollisessa käytössä myös lääkinnällisen laitteen automaatioprosessi tulisi ke-

hittää kuin lääkinnällinen laite. Tässä tutkielmassa emme kuitenkaan käy läpi, miten jatkuvan integraation ja tuotantoonviennin automaation muutoshallinta tulisi järjestää, jotta se täyttäisi lääkinnällisen laitteen vaatimukset.

Luvussa 2 käsittelemme lääkinnällisiä laitteita, sekä niihin kohdistuvaa regulaatiota. Käsittelemme erityisesti ohjelmistopohjaisten lääkinnällisten laitteiden regulointia ja MDR vaikutusta niiden kehitykseen. Luvussa 3 käymme läpi mitä DevOps ja jatkuva tuotantoonvienti sekä integraatio tarkoittavat. Avaamme myös ketterien kehitysmallien yhtäläisyyksiä. Luku 4 käsittelee laatuvaatimuksia lääkinnällisille laitteille, joka toimii alustuksena luvulle 5, jossa paneudumme syvälle ohjelmistopohjaisten lääkinnällisten laitteiden kehityksen vaatimuksiin, sekä osoitamme niiden automaation mahdollisuudet. Luku 6 esittelee johtopäätökset ja on yhteenveto tämän työn sisällöstä.

2 Lääkinnälliset laitteet ja niiden regulaatio

Tässä luvussa käymme läpi mitä lääkinnälliset laitteet ovat, miten niitä luokitellaan ja millaista regulaatiota niihin kohdistetaan Euroopan Unionin tasolla. Vastaamme tässä luvussa myös tukimuskysymyksiin RQ1 ja RQ2, sillä käymme läpi kaikki lääkinnällisten laitteiden luokittelut, sekä MDR:n (engl. Medical Device Regulation) vaikutukset lääkinnällisten laitteiden luokitteluun.

2.1 Määritelmä

Lääkinnällinen laite on mikä tahansa instrumentti, laitteisto, väine, koje, laite, implantti, in vitro -reagenssi, *ohjelmisto*, materiaali tai muu samankaltainen tai liittyvä tarvike, jolla on tarkoitus valmistajan määritelmän mukaan tehdä mm. sairauden diagnosointia, ehkäisyä, tarkkailua, hoitoa tai lievitystä (ISO 13485, 2016).

ISO 13485-standardi ottaa myös laajemmin kantaa käyttötarkoituksista joista jonkun tulee täyttyä, jotta laite voidaan määritellä lääkinnälliseksi laitteeksi. Näitä käyttötarkoituksia on paljon, joista muutamana nostona voidaan mainita sairauksien diagnosoinnin, ehkäisyn, tarkkailun, hoidon ja lievityksen lisäksi mm. hedelmöittymisen säätely, sekä elintoimintojen tukeminen ja ylläpito.

Ohjelmistopohjaisilla lääkinnällisillä laitteilla tarkoitetaan lääkinnällisiä laitteita, joilla ei ole fyysistä muotoa, kuten stetoskoopilla tai sydänkeuhkokoneella. Euroopan komissio käyttää tähän viittaavaa termiä *Medical Device Software (MDSW)*, mutta ei rajoita sen riippumattomuutta jostain fyysisestä laitteesta. Tässä tutkielmassa käytämme kuitenkin termiä *ohjelmistopohjainen lääkinnällinen laite* siinä tarkoituksessa, ettei sillä ole laittelle ominaista fyysistä olomuotoa, eikä se ole sulautettu ohjelmisto fyysiselle lääkinnälliselle laitteelle.

Lääkinnällisiksi laitteiksi ei lasketa kuitenkaan kaikkia ohjelmistoja, joissa esimerkiksi potilastietoa käsitellään. Tällaisia ohjelmistoja voivat olla esimerkiksi potilastietojärjestelmät, jotka mielletään vain tiedon tallentamista ja näyttämistä varten raken-

netuiksi ohjelmistoiksi. Tällainen ohjelmisto, joka ei tee omaa analyysiä vaan puhtaasti tallentaa ja näyttää tallennettua dataa ei ole lääkinnällinen laite tai ohjelmistopohjainen lääkinnällinen laite. Potilastietojärjestelmät nähdään enemmänkin paperin korvaajana kuin laitteena, joka voisi vaikuttaa potilaan hoitopäätöksiin (MDR Guidance, European Commission, 2019).

2.2 Lääkinnällisten laitteiden luokittelu

Valmistaja luokittelee lääkinnällisen laitteen sen tuottaman riskin mukaisesti. Riski määritellään sen mukaisesti millaista potentiaalista haittaa lääkinnällinen laite voi aiheuttaa potilaalle ja mikä sen esiintymistodennäköisyys on (IEC 62304, 2015).

Lääkinnällisten laitteiden luokitteluun on käytössä useita järjestelmiä (Food ja Administration, 2018) (Medical Device Regulation, European Union, 2017). Tässä tutkielmassa keskitytään Euroopan Unionin tapaan tehdä lääkinnälisten laitteiden luokitus. Luokittelussa riski potilaalle määrittää luokan, johon lääkinnällinen laite kuuluu. Mahdollisia luokkia ovat 1, 2a, 2b ja 3. (Medical Device Regulation, European Union, 2017). Tutkielmassa käytämme pohjana myös IEC 62304-standardin määrittämiä vaatimuksia lääkinnällisille laitteille, IEC 62304-standardin mukainen luokittelu on A, B ja C. Nämä luokitukset eivät linkity suoraan EU-lainsäädännössä käytettyyn luokitteluun, vaikkakin taulukko 2.1 on eräänlainen tapa linkittää IEC 62304 ja EU-lainsäädäntö yhteen. On kuitenkin teoreettisesti mahdollista, että lääkinnällinen laite on IEC 62304 luokassa C, mutta EU lainsäädännön mukaan luokassa 2a. Linkitykseen palaamme vielä alaluvussa 2.3.

IEC 62304-standardissa käytetään luokittelua A, B, C, jossa A on pienimmän riskin laite ja C suurimman riskin laite (IEC 62304, 2015). Luokitusten vaatimuksien yhteensovitusta käydään läpi tämän luvun alakohdissa.

2.3 Lääkinnällisten laitteiden luokittelu eri standardien ja lainsäädäntöjen mukaan

Kuva 2.1 avaa, miten Medical Device Regulation (MDR) luokittelee lääkinnälliset laitteet eri luokkiin. Luokitus lähtee kuitenkin liikkeelle siitä, että onko kyseessä lääkinnällinen laite vai ei. Kuten aiemmin todettu, mm. potilastietojärjestelmät, jot-

	IEC	MDR	Perustelu
Vähäinen riski potilaalle	A	2a	MDR:ssä luokka 2a ei voi sisältää laitteita, jotka johtaisivat vakavaan haittaan tai kuolemaan, joita IEC luokitukset B ja C sisältävät
Riski voi aiheuttaa realisoituessaan terveydentilan huonontumisen	B	2b	MDR:n 2b mainitaan vakava terveydentilan huonontuminen, joka voi olla IEC C myös, pääsääntöisesti kuitenkin esimerkiksi monitorit ovat luokassa 2b ja IEC B
Riski voi aiheuttaa realisoituessaan vakavan terveydentilan huonontumisen tai kuoleman	C	3	Sekä MDR, että IEC viittaavat luokilla 3 ja C kuolemaan ja vakavaan vammautumiseen

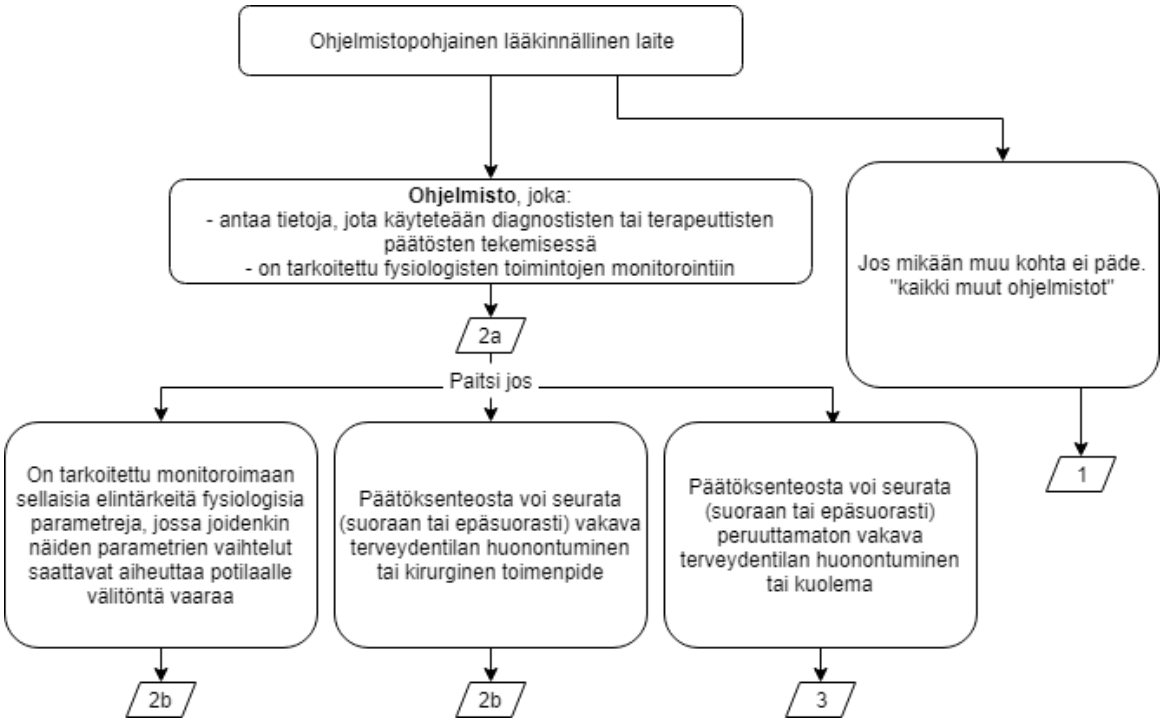
Taulukko 2.1: Eri standardien ja lainsäädäntöjen linkittyvyys toisiinsa

ka vain toimivat paperin korvikkeena tai arkistona, eivät täytä lääkinnällisen laitteen määritelmää. Kuvassa 2.2 taas näemme miten IEC 62304-standardi luokittelee ohjelmistopohjaiset lääkinnälliset laitteet eri luokkiin.

Vaikka luokittelumetodeita on useita, on niillä kaikilla samankaltainen skaala alkaen riskistä, joka realisoituessaan ei voi erityisemmin vahingoittaa potilasta ja päättyen riskiin, joka realisoituessaan voi olla hengenvaarallinen. Taulukko 2.1 avaa miten tässä tutkielmassa luokittelut eri asteikoiden mukaan voidaan tietyllä tasolla kytkeä toisiinsa. Taulukosta on tiputettu pois MDR:n luokka 1, sillä EU komission ohjelmistopohjaisten lääkinnällisten laitteiden tulkintaohjeen mukaisesti ohjelmistopohjainen lääkinnällinen laite ei käytännössä voi olla kyseisessä luokassa. (MDR Guidance, European Commission, 2019).

2.3.1 Luokan 1 lääkinnällinen laite

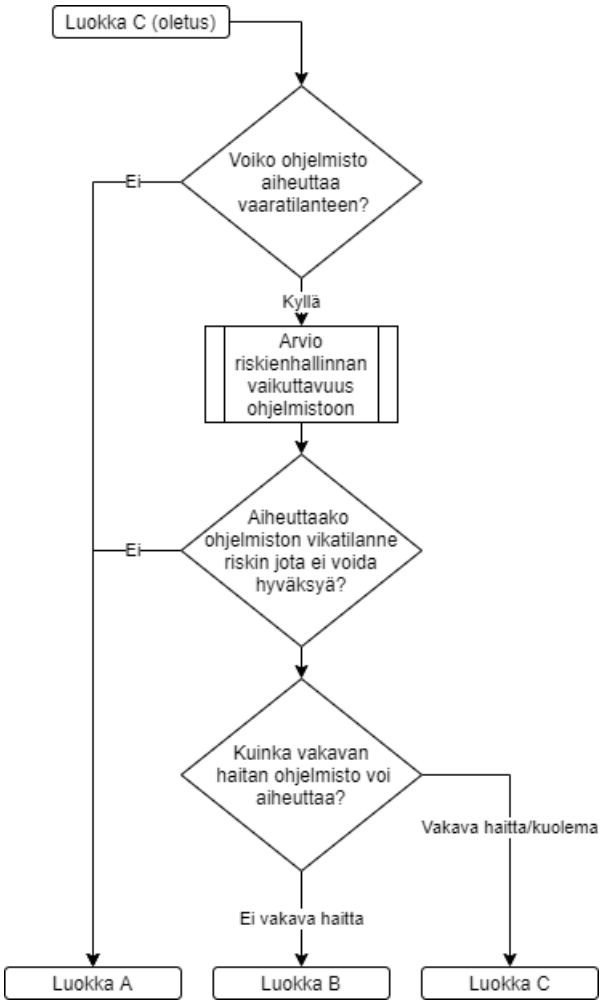
Emme tässä tutkielmassa käsittele luokan 1 lääkinnällisiä laitteita, sillä Euroopan komission MDR lainsäädännön tulkintaohjeen mukaisesti ohjelmistopohjaisia



Kuva 2.1: Lääkinnällisten laitteiden luokittelu (Medical Device Regulation, European Union, 2017)

lääkinnällisiä laitteita ei voi olla luokassa 1, vaikka itse lainsäädäntö mahdollistaisi ”muut ohjelmistot”, jotka voisivat kuulua luokkaan 1 (MDR Guidance, European Commission, 2019). Näin ollen ohjelmistopohjaisten lääkinnällisten laitteiden näkökulmasta luokan 1 laitteet eivät ole relevantteja. Mikäli ei puhuta ohjelmistopohjaisista lääkinnällisistä laitteista, niin luokkaan 1 kuuluu hyvin matalan riskin laitteita, kuten pyörätuoleja ja laastareita.

Tutkimuskysymys **RQ1: Mitkä ovat luokkien 1 ja 2a lääkinnällisten laitteiden erot?** osoittautuu näin ollen MDR:n jälkeen ohjelmistojen osalta epärelevantiksi, sillä luokkaan 1 ei kuulu ohjelmistopohjaisia lääkinnällisiä laitteita ja täten ohjelmistojen osalta eroja ei voida vertailla. Sen sijaan MDD:n aikakaudella luokkien 1 ja 2a erot olivat erityisesti riskeissä; luokan 2a lääkinnällisten laitteiden riskit olivat korkeampia kuin luokan 1 (Medical Device Directive, European Union, 1993). Tämän lisäksi MDD aikakauden luokan 1 lääkinnällisiä laitteita ei ole auditoitu ilmoitetun laitoksen toimesta, vaan näille laitteille riitti itseilmoitus viranomaiselle – näin ollen varianssi luokan 1 ja 2a lääkinnällisten laitteiden laadussa ja ohjelmistokehitysprosesseissa on oletettavasti ollut isompi.



Kuva 2.2: Lääkinnällisten laitteiden luokittelu (IEC 62304, 2015)

2.3.2 Luokan 2 lääkinnällinen laite

MDR:n mukaisesti luokka 2 jakautuu kahteen eri osaan: 2a ja 2b, joista 2a on matalamman riskin laite ja 2b korkeamman riskin laite (Medical Device Regulation, European Union, 2017). Suurin osa ohjelmistopohjaisista lääkinnällisistä laitteista, jotka kuuluivat MDD:n aikaan luokkaan 1, siirtyvät MDR:n myötä luokkaan 2a johtuen alimman riskitason noususta MDR:n luokittelusäännön 11 vuoksi.

Luokan 2a ja 2b ohjelmistopohjaisia laitteita ovat käytännössä kaikki ohjelmistot paitsi ne, jotka ovat tarkoitettu monitoroimaan elintärkeitä fysiologisia parametrejä, joissa virhetilanne voisi aiheuttaa potilaalle vaaraa, tai päätöksenteosta voisi seurata vakava terveydentilan huonontuminen, kirurginen toimenpide, peruuttamaton vakava terveydentilan huononeminen tai kuolema (Medical Device Regulation, European Union,

2017).

Luokan 2b ohjelmistopohjaiset laitteet voivat aiheuttaa virheellisesti toimiessaan potilaalle vaaraa, mutta eivät peruuttamatonta ja vakavaa terveydentilan huonontumista tai kuolemaa.

2.3.3 Luokan 3 lääkinnällinen laite

Luokan 3 ohjelmistopohjaiset lääkinnälliset laitteet ovat suurimman potentiaalisen riskin omaavia laitteita, joiden virhetilanne voi suoraan tai epäsuorasti aiheuttaa peruuttamattoman ja vakavan terveydentilan huonontumisen tai jopa kuoleman (Medical Device Regulation, European Union, 2017). Esimerkiksi ohjelmisto, joka tekee automaattisesti hoitopäätöksen akuutista aivohalvauksesta kuva-analyysin perusteella, luokitellaan luokkaan 3 (MDR Guidance, European Commission, 2019).

2.4 Medical Device Regulation (MDR) kautta tulevat uudet vaatimukset

Ennen MDR:n aikakautta Euroopan Unionin alueella on ollut voimassa MDD (Medical Device Directive), jonka piirissä merkittävä osa itsenäisistä ohjelmistopohjaisista lääkinnällisistä laitteista on voitu luokitellan luokkaan 1, jossa riittää pelkkä itseilmoitus viranomaiselle (Medical Device Directive, European Union, 1993).

MDR:n myötä tämä tulkinta tiukkenee merkittävästi, sillä MDR lainsäädäntö (Sääntö 11) sanoo suoraan, että kaikki ohjelmistot, jotka on tarkoitettu antamaan tietoja, joita hyödynnetään diagnostisten tai terapeuttisten päätösten tekemisessä, luokitellaan luokkaan 2a, paitsi jos nämä päätökset voivat aiheuttaa suuremman riskin, jolloin luokka voi olla 2b tai 3. Muussa tapauksessa ohjelmistopohjainen lääkinnällinen laite kuuluu luokkaan 1 (Medical Device Regulation, European Union, 2017).

Näin ollen käytännössä kaikki ohjelmistopohjaiset lääkinnälliset laitteet, joita käytetään sairauden hoitoon, ehkäisemiseen tai diagnostisten päätösten tekoon luokitellaan automaattisesti vähintään luokkaan 2a, kuten voidaan todeta mm. MDR:n säännön 11 tulkintaohjeesta, jossa myös pienimmän riskin ohjelmistopohjaiset lääkinnälliset laitteet kuuluvat luokkaan 2a (MDR Guidance, European Commission, 2019).

Tämän työn yksi keskeisistä tutkimuskysymyksistä oli nimenomaan **RQ2: Miksi ohjelmistopohjaiset lääkinnälliset laitteet ovat EU:n uuden lainsäädännön mukaisesti lähtökohtaisesti luokassa 2a, eivätkä luokassa 1?**. Tähän kysymykseen vastaa itse lainsäädäntö suhteellisen suorasti, sillä yksikään ohjelmistopohjainen lääkinnällinen laite ei käytännössä voi olla alemmassa luokassa kuin 2a (MDR Guidance, European Commission, 2019). Ennen MDR-lainsäädäntöä MDD-aikakaudella ohjelmistopohjaiset lääkinnälliset laitteet, joiden riski oli pieni, pystyivät olemaan luokassa 1 (Medical Device Directive, European Union, 1993). **MDR siis muuttaa kaikki MDD:n aikaiset luokan 1 ohjelmistopohjaiset lääkinnälliset laitteet vähintään MDR:n mukaiseen luokkaan 2a**.

MDR:n myötä tulee myös paljon muita muutoksia lääkinnällisiin laitteisiin, mutta tämän tutkielman kannalta merkittävin muutos tulee nimenomaan ohjelmistopohjaisten lääkinnällisten laitteiden luokittelun muutoksen kautta.

3 DevOps, jatkuva integraatio sekä tuotantoonvienti

Tässä luvussa kuvaamme, mitä DevOps tarkoittaa ja miten jatkuva integraatio sekä tuotantoonvienti toimivat DevOpsin tärkeinä tukipilareina.

3.1 Jatkuva integraatio

Jatkuva integraatio (engl. Continuous Integration, CI) tarkoittaa toimintatapaa, jota tukee valikoima työkaluja ja ohjelmistoja, jossa sovelluskehitystiimin tuottamat muutokset versiohallintaan verifioidaan mahdollisimman nopeasti ja automaattisesti, että ne ovat hyviä - eivätkä esimerkiksi riko ohjelmistoa tai ohjelmistokappaletta (Meyer, 2014).

Jatkuva integraatio on tärkeä toimintatapa DevOps-mallilla toimivassa kehityksessä, sillä nopean kehityssyklin edellytyksenä on nopea palaute muutoksista (Smeds et al., 2015).

Jatkuvan integraation prosessi on seuraava: ohjelmistokehittäjä vie versiohallintaan haluamansa muutoksen, jolloin jatkuvaan integraatioon erikoistunut palvelin hakee muutokset, ajaa valikoiman testejä, analyysejä ja muita työkaluja, joiden lopputuloksena syntyy raportteja sekä tieto siitä, onnistuiko paketointi (engl. Build). Näin ollen ohjelmistokehittäjä saa nopeasti tiedon siitä, onko hänen tekemänsä muutos sellainen, että se voidaan viedä tuotantoon asti ilman, että se rikkoo mitään olemassa olevaa. (Meyer, 2014)

Erilaisia jatkuvan integraation palvelimia on useita ja nykyään ne ovat useammin integroitu osaksi esim. versiohallinnan kokonaisuutta, kuten esimerkiksi Bitbucket Pipelines, joka toimii osana Bitbucket versiohallintaa (*Bitbucket Pipelines* 2019).

Jatkuva integraatio pystyy muutosten verifioinnin lisäksi ajamaan myös paljon muita työkaluja. Yksi esimerkki on dokumentaation generointi, jota käytämme myöhemmin tässä työssä lääkinnällisen laitteen teknisen tiedoston muodostamiseen esimerkkitapauksessa alaluvussa 5.4.

3.2 Jatkuva tuotantoonvienti

Jatkuvalla tuotantoonviennillä (engl. Continuous Deployment, CD) tarkoitetaan nimen mukaisesti ohjelmiston jatkuvaa vientiä tuotantoon (tai muihin ympäristöihin). Jatkuva tuotantoonvienti ja jatkuva integraatio toimivat usein käsi kädessä; heti sen jälkeen kun jatkuva integraatio on varmistanut, että kaikki muutokset ovat valideja sekä suorittanut muut vaaditut toimenpiteet, jatkuva tuotantoonvienti vie muutokset tuotantoympäristöön (Leppänen et al., 2015).

Jatkuva tuotantoonvienti tarkoittaa siis ohjelmiston automatisoitua asennusta tuotannon ympäristöön. Tähän kuuluu useita osuuksia, joista yksi on itse ohjelmiston asennus, mutta usein jatkuvalla tuotantoonviennillä voidaan hallita myös esimerkiksi infrastruktuurin konfiguraatiota.

Tärkeimpinä hyötyinä jatkuvalle tuotantoonviennille, niin kuin koko DevOps:lle, on nopea palaute loppukäyttäjiltä, jotka itse tuotetta käyttävät (Leppänen et al., 2015). Tämän lisäksi ohjelmistokehitystsiimi voi itse hallita suoraan, milloin he vievät asioita tuotantoon tai ottavat niitä pois tuotannosta.

Jatkuva tuotantoonvienti ei tarkoita samaa asiaa kuin automaattinen tuotantoonvienti. Automaattinen tuotantoonvienti jokaisesta onnistuneesta muutoksesta on triviaalia toteuttaa jatkuvan tuotantoonviennin kyvykkyyksien rakentamisen jälkeen (Leppänen et al., 2015).

3.3 Johdanto DevOps:iin

DevOps:n tarkoitus on ensisijaisesti pienentää eroa kehityksen ja tuotannon välillä, näin ollen nimikin on muodostunut kahdesta osasta: Dev (englannin kielen sana Development, jolla viitataan kehitykseen) ja Ops (englannin kielen sana Operations, jolla viitataan tuotantoon) (Smeds et al., 2015). Smeds et al. määrittelee DevOps:n olevan valikoima tietotekniikan prosessin kyvykkyyksiä, joita tuetaan tietyillä kulttuurisilla ja teknologisilla mahdollistajilla. Tässä työssä keskitymme erityisesti teknologisiin mahdollistajiin, joihin lasketaan jatkuva integraation ja jatkuva tuotantoonvienti. Emme syvenny kulturaalisiin vaikutuksiin ja vaatimuksiin organisaatiossa työn tutkimuskysymysten rajauksen vuoksi.

DevOps-prosessin vaiheita ovat määrittely, kehitys, testaus ja julkai-

su/tuotantoonvienti (Smeds et al., 2015). Tärkeää on kytkeä nämä aktiviteetit toimimaan jatkuvasti niin, että palaute päätyy nopeasti prosessin lopusta takaisin alkuun. Tämä vaatii sen, että prosessin sykli on nopea, eikä katkoksia synny eri vaiheiden välillä.

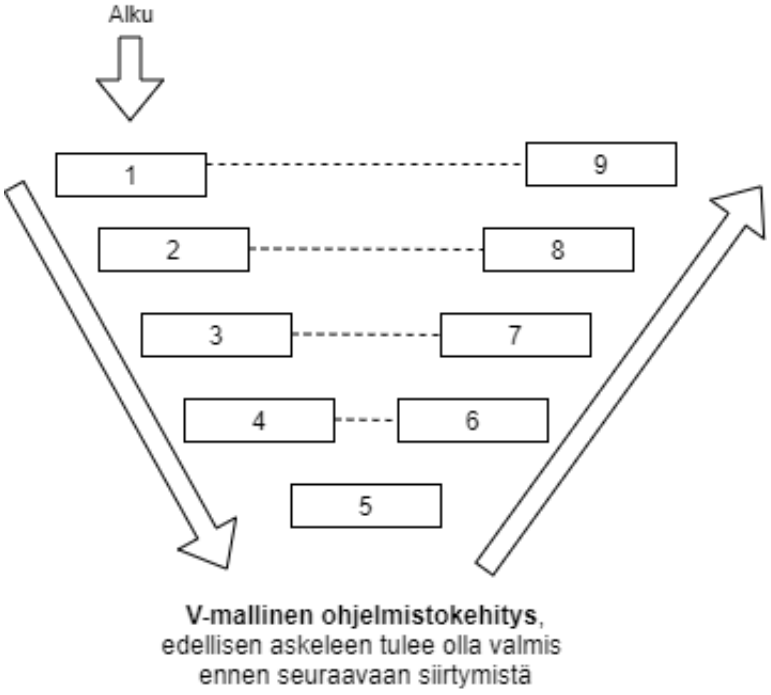
Prosessin nopean syklin saavuttaminen vaatii usein tuekseen teknisiä kyvykkyyksiä, joita ovat jatkuva suunnittelu, osallistava ja jatkuva kehitys, jatkuva integraatio ja testaus, katkuva julkaisu ja tuotantoonvienti, jatkuva infrastruktuurin monitorointi ja optimointi, jatkuva käyttäjien toiminnan monitorointi ja palautteensaanti sekä välitön virheistä toipuminen ilman viiveitä (Smeds et al., 2015). Näistä keskitymme erityisesti jatkuvaan integraatioon ja tuotantoonvientiin, jonka ympärille myös muut osat teknisistä kyvykkyyksistä osittain kytkeytyvät. Teknologisia mahdollistajia jatkuvalle integraatiolle ja julkaisulle ovat automaattinen testaus, paketointi (engl. Build), tuotantoonvienti sekä infrastruktuurin hallinta (Smeds et al., 2015).

Ohjelmistopohjaisten lääkinnällisten laitteiden kehityksessä, kuten muidenkin reguloitujen ohjelmistokehitysprosessien, on usein totuttu vesiputousmalliseen kehitykseen, jolloin koko kehitys on käytännössä yksi sykli, jossa vaihe vaiheelta siirrytään eteenpäin. Vaiheet voivat olla merkittävänkin pitkiä, ja niitä ei toisteta välttämättä kuin kerran. Näin ollen automaatiolle ei ole niin kovaa tarvetta. DevOps-malliin kuuluu oleellisesti se, että prosessin vaiheet toistuvat useita kertoja syklisesti, kuitenkin niin että käytännössä samat vaiheet kuin vesiputousmallisessa kehityksessä tulevat käytyä läpi (Laukkarinen et al., 2017).

3.4 DevOps ja V-malli yhdessä

V-malli on ohjelmiston kehitysmalli, joka toimii lineaarisesti. Se on piirrettynä prosessina nimensä mukaisesti V:n muotoinen – vasemmassa yläkulmassa tehdyt määrittelyt verifioidaan oikeassa yläkulmassa. Prosessi lähtee ylätason määrittelyistä, jonka jälkeen määrittelyitä tarkennetaan. Lopulta päädytään toteutukseen ja tämän jälkeen jokaista määrittelyvaihetta vasten tehdään verifiointi siitä, että vastaako lopputulos aiemmin määritettyä (Balaji ja Murugaiyan, 2012). Kuva 3.1 havainnollistaa vielä prosessin kulkua, sekä verifioinnin ja määrittelyn suhdetta toisiinsa eri tasoilla.

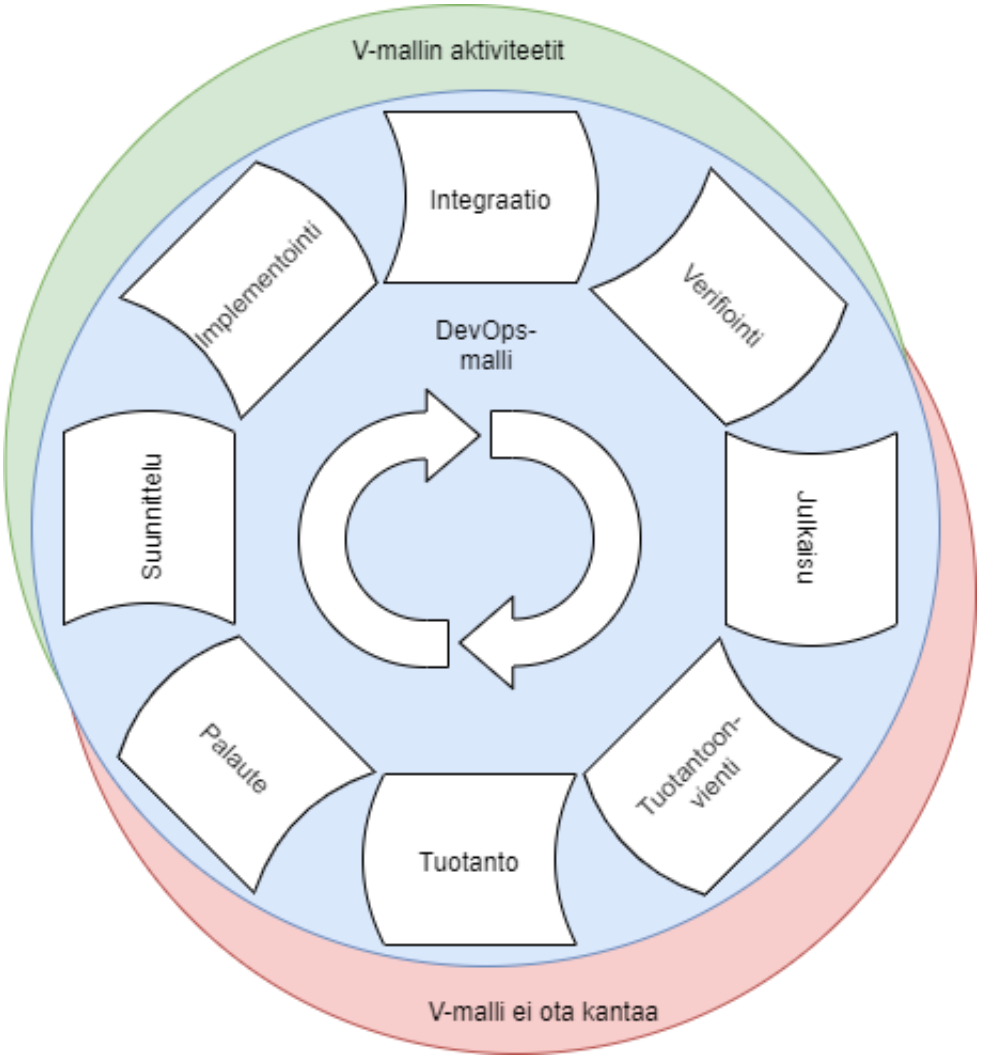
DevOps- ja V-mallinen ohjelmistokehitys eivät ole toistensa vastakohtia, vaan V-malli voidaan sijoittaa osaksi DevOps-tyylistä toimintatapaa, kuten kuva 3.2 osoittaa. Näin ollen saamme luotua iteratiivisen V-mallin kaltaisen ohjelmistokehitysmallin, jossa oh-



Kuva 3.1: V-malli

jelmistokehityksen eri aktiviteetteja tehdään syklissä peräkkäin.

V-mallinen ohjelmistokehitysprosessi ei ota laisinkaan kantaa siihen, miten tuotanto tai ohjelmiston vienti tuotantoon järjestetään, mikä käytännössä tarkoittaa DevOpsin Ops puolta. Dev puoli voidaan toteuttaa V-mallilla, kunhan sykli on vain tarpeeksi nopea. Tämä tarkoittaa, että V-mallin osuuteen ei voida viedä isoja muutoksia, vaan isot muutokset pitää palastella niin, että ne voidaan suorittaa nopealla kehityssyklillä läpi DevOps-prosessin.



Kuva 3.2: DevOps ja V-malli

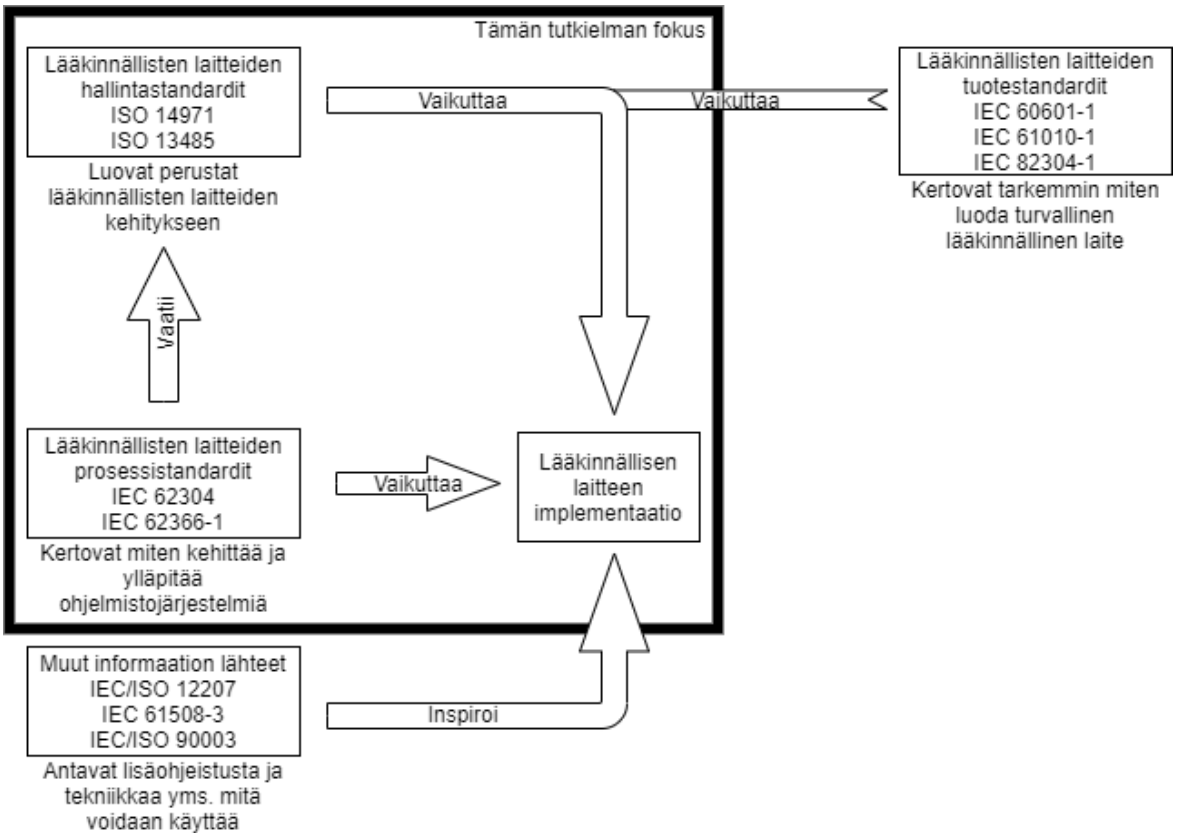
4 Laatuvaatimukset lääkinnällisille laitteille

Käymme tässä luvussa läpi mitkä kaikki tekijät, kuten regulaatio ja standardit, vaikuttavat lääkinnällisten laitteiden kehitykseen. Käsittelemme erityisesti ISO 13485- ja IEC 62304-standardeja, jotka ovat tämän tutkielman kannalta merkityksellisimmät. Käymme myös läpi yleisellä tasolla muut standardit, jotka lääkinnällisen laitteen kehityksessä on otettava huomioon.

4.1 Johdanto lääkinnällisten laitteiden laatuun

Lääkinnällisten laitteiden kehittäminen, oli sitten kyseessä ohjelmistopohjainen tai laitteistopohjainen lääkinnällinen laite, vaatii aina erilaisia laatuprosesseja ympärilleen (Medical Device Regulation, European Union, 2017). Näitä laatuprosesseja on kuvattu useissa monikansallisissa standardeissa, jotka keskittyvät nimenomaan lääkinnällisten laitteiden standardisointiin. Tämän lisäksi on olemassa kansallista ja EU tasoista regulaatiota sekä ohjeistusta, jotka tulee kehityksessä ottaa huomioon (IEC 62304, 2015). Näistä eniten huomioon otettavat ovat standardit ja lainsäädännöt ovat ISO 13485, IEC 62304, ISO 14971, IEC 62366-1 ja EU:n Medical Device Regulation (MDR).

Vaikka standardit kuten IEC 62304 eivät ota kantaa käytettävään ohelmistokehitysmetodiin, on suurin osa aktiviteeteista kuvattu kuitenkin niin, että niiden toteuttaminen on helpointa vesiputousmallisessa kehityksessä (IEC 62304, 2015) (Laukkarinen et al., 2017). IEC 62304-standardissa esitelty esimerkkiohjelmistokehitysprosessi nojaa vahvasti malliin, jossa ohjelmisto ensin suunnitellaan huolellisesti, jonka jälkeen tehdään toteutus ja lopuksi verifiointi sen suhteen, saatiinko sitä mitä haluttiin. Jatkuva integraatio (Continous Integration) ja jatkuva tuotantoonvienti (Continous Delivery) eivät itsessään vielä suoraan tarkoita ohjelmistokehitysmallia, mutta niiden hyödyt tulevat siitä, mitä tiheämmällä kierrolla ohjelmistoa kehitetään ja viedään tuotantoon. On siis mahdollista toteuttaa CI- ja CD-automaatio myös vesiputousmalliseen kehitykseen ja käyttää niitä hyvin harvoin. Tällöin hyöty automaatiosta suhteessa siihen käytettyihin resursseihin jää pienemmäksi. Ketterä ohjelmistokehitys, CI ja CD toimivat suhteessa



Kuva 4.1: Standardien suhde toisiinsa (IEC 62304, 2015)

panostuksiin erittäin hyvin. Tällä tähdätään siihen, että tiheässä inkrementaalissa kehityksessä tapahtuvat manuaaliset toimenpiteet saadaan automatisoitua täysin. Tällöin prosessi keskittyy vain uuden arvon tuottamiseen ja manuaaliset askeleet ovat täysin automatisoituja (Laukkarinen et al., 2017).

4.2 Laatujärjestelmä ja riskienhallinta reguloidussa kehityksessä (ISO 13485)

ISO 13485-standardi määrittää laatujärjestelmän, joka pyrkii takaamaan laadukkaiden lääkinnällisten laitteiden kehityksen. Se ei teknisellä tasolla ota kantaa siihen, miten esimerkiksi ohjelmistoja tulisi kehittää. Se kattaa lääkinnälliset laitteet vaikka niissä ei sovelluskomponenttia edes olisi, kuten stetoskoopin. Standardi keskittyy lääkinnällisten laitteiden suunnittelun, kehityksen, ostamisen, tuotantoonviennin, monitoroinnin ja parannusten ympärillä toimiviin prosesseihin. Se ei suoraan ota kantaa, miten laatuprosessit tulisi implementoida. Standardi ottaa kantaa myös sellaisiin asioi-

hin, joilla ei tämän tutkielman kannalta ole merkitystä, kuten yritysjohdon sitoutumiseen lääkinnällisten laitteiden kehitykseen (ISO 13485, 2016).

4.3 Lääkinnällisten laitteiden ohjelmiston elinkaaren prosessit (IEC 62304)

IEC 62304-standardi määrittää lääkinnällisten laitteiden ohjelmistoihin liittyvät elinkaaren prosessit alkaen suunnittelusta ja jatkuen julkaisun jälkeen ylläpitoon. Standardia ei ole kirjoitettu ohjelmistopohjaisille lääkinnällisille laitteille, vaan se ottaa kantaa yleisesti lääkinnällisissä laitteissa käytettävän ohjelmiston prosesseihin. Tämä usein tarkoittaa sitä, että ohjelmisto on suunniteltu käytettäväksi yhdessä jonkin fyysisen laitteen kanssa, jossa ohjelmisto on yhdessä osassa. Lääkinnällisen laitteen ollessa täysin ohjelmistopohjainen on siihen helpompi soveltaa CI- ja CD-automaatioita, sillä sulautettujen järjestelmien tuotantoonviennin automaatioon liittyy merkittävästi enemmän haasteita (Lwakatare et al., 2016). Standardin pääkohdat ovat ohjelmistokehityksen ja -ylläpidon prosessien vaatimusten määrittelyssä, riskienhallinnassa, konfiguraation hallinnassa, sekä ongelmienratkaisussa (IEC 62304, 2015).

4.4 Muut relevantit standardit lääkinnällisten laitteiden kehityksessä

Käsittelemme tässä alaluvussa standardeja, jotka vaikuttavat ohjelmistopohjaisten lääkinnällisten laitteiden kehittämiseen, mutta joihin ei tässä tutkielmassa syvennytä sen enempää.

Riskinhallinnan soveltaminen ohjelmistopohjaisten lääkinnällisten laitteiden kehitykseen (ISO 14971) ISO 14971-standardi painottuu nimensä mukaisesti erityisesti siihen, miten riskinhallintaa sovelletaan lääkinnällisiin laitteisiin. Se ottaa kantaa erityisesti riskin kahteen pääosa-alueeseen eli ilmenemistodennäköisyyteen ja seurausten vakavuuteen (ISO 14971, 2009). Emme tässä tutkielmassa käy tämän tarkemmin läpi, mitä vaatimuksia esimerkiksi suunnitteluun ISO 14971 tuo.

Suunnittelun laatuvaatimukset (IEC 62366) IEC 62366-standardi kuvaa tavan, miten käytettävyys otetaan huomioon kehitettäessä lääkinnällisiä laitteita (IEC

62366, 2015). Se auttaa lääkinnällisten laitteiden kehittäjiä saavuttamaan korkean käytettävyyden tason laitteissa ja näin vähentämään riskejä, jotka aiheutuvat mm. väärän tyylisestä käyttötavasta lääkinnälliselle laitteelle (IEC 62366, 2015).

4.5 Jäljitettävyys

Lääkinnällisten laitteiden kehityksessä erityisesti jäljitettävyys on tärkeää (ISO 13485, 2016). Kaikkien muutosten tulee olla jäljitettävissä käytännössä alkaen siitä hetkestä, kun niistä ensimmäisen kerran puhutaan. Eri standardit tarkentavat omalta osaltaan, mitä asioita pitää pystyä jäljittämään. Esimerkiksi IEC 62304-standardi ottaa kantaa, miten ohjelmiston eri muutosten tulee olla jäljitettäviä muutospyynnöstä tuotantoon asti. Jäljitettävyydellä tarkoitetaan kykyä yhdistää prosessin eri vaiheita yhteen (kuten määrittely, toteutus, verifiointi ja tuotantoonvienti) (IEC 62304, 2015).

4.6 SOUP - Software Of Unknown Provenance ja sen ongelmatiikka

SOUP (engl. Software of Unknown Provenance, myös Software of Unknown Pedigree) tarkoittaa ohjelmistokappaletta, jota ei alunperin ole kehitetty lääkinnällisen laitteen vaatimien kriteereiden mukaisesti. IEC 62304 mukaisen riskiluokittelun mukaisesti SOUP ohjelmistokappaleiden käyttö edellyttää erityisiä hallintakeinoja, mikäli lääkinnällisen laitteen riskiluokka on B tai C (A riskiluokan lääkinnälliset laitteet voivat käyttää SOUP ohjelmistokappaleita IEC 62304 näkökulmasta suoraviivaisemmin). Jotta SOUP ohjelmistokappaleita voidaan ottaa käyttöön lääkinnälliseen laitteeseen, tulee valmistajan varmistua niiden käyttökelpoisuudesta ja kantaa vastuu niidenkin aiheuttamista mahdollisista häiriöistä. Käytännössä tämä tarkoittaa, että jokainen SOUP ohjelmistokappale tulee käydä läpi, testata ja sille tulee tehdä riskiarvio ja harkinta voidaanko sitä käyttää (IEC 62304, 2015).

Nykyaikaisessa JavaScript-pohjaisessa sovelluksessa, jossa on käytössä npm (Node Package Manager), SOUP-ohjelmistokappaleiden määrä voi olla tuhansia jo yksinkertaisellekin SPA (Single Page App) sovellukselle. Alaluvussa 5.4 esiteltävässä esimerkkisovelluksessa käytetty React Framework toi riippuvuuksien kautta 998 SOUP-ohjelmistokappaletta (React-asennus tehty 9.11.2019 versiolla 16.11.0). Käytännössä

tällaisen ohjelmistokappalemäärän hallinta ja automaattitestaus on mahdotonta tai vähintäänkin niin hidasta, että sillä rampautetaan ohjelmistokehityksen julkaisusyklin nopeus.

Eräässä tutkimuksessa, joka keskittyy nimenomaan turvallisuuskriittisiin järjestelmiin, on esitetty riskienarviointikehystä, jonka puitteissa SOUP ohjelmistokappaleita voitaisi arvioida helpommin ja tätä kautta perustella SOUP ohjelmistokappaleiden käyttöä turvallisuuskriittisissä järjestelmissä – kuten lääkinnällisissä laitteissa (Cook et al., 2015). Tämä mahdollistaisi esimerkiksi React Frameworkin kaikkien SOUP-ohjelmistokappaleiden hallinnan yhdessä, jolla on merkittävä vaikutus JavaScript maailman lääkinnällisten laitteiden ohjelmistokehityksen ketteryyteen.

Muita mahdollisuuksia voivat olla mm. suuren SOUP-komponenttimäärän käsitteleminen kokonaisuutena, näin voitaisi toimia esimerkiksi npm-pakettien kanssa, missä riippuvuuksien kautta tulevien pakettien määrä on usein suuri. Tällöin IEC 62304:n osassa SOUP vaatimuksia toiminnalliset- ja performanssivaatimukset tehtäisiin kokonaisuudelle, eikä jokaiselle SOUP-komponentille erikseen. Tämä nopeuttaa työtä huomattavasti.

5 Jatkuva integraatio ja toimittaminen ohjelmistopohjaisten luokan 2a lääkinnällisten laitteiden kanssa

Tässä luvussa käymme läpi, miten jatkuva integraatio ja toimittaminen ohjelmistopohjaisten lääkinnällisten laitteiden kanssa voidaan toteuttaa. Käsittelemme tässä luvussa teknisen kontribuution kautta RQ3 ”Voidaanko luokan 2a ohjelmistopohjaisen lääkinnällisen laitteen jatkuva integraatio ja tuotantoonvienti automatisoida ja miten?”.

Tämä luku alkaa korkeammalta abstraktiotasolta käsitellen ohjelmistokehitykseen tulevia vaatimuksia regulaatiosta, mutta sukeltaa loppua kohden syvälle yksityiskohtiin ja esimerkkeihin siitä, miten automaatio voi auttaa ja nopeuttaa kehitysprosessia, johon kohdistuu regulatoorisia vaatimuksia.

5.1 Johdanto lääkinnällisten laitteden jatkuvaan integraatioon ja toimittamiseen

Kuten jo aiemmin todettu, on lääkinnällisten laitteiden kehitysmallit suunniteltu vesiputousmaiseksi, jossa kehityksen seuraavaan askeleeseen siirrytään edellisen valmistuttua. Tämä selviää jo katsomalla standardeja, joissa ohjelmistokehitysprosessi kuvataan (IEC 62304). Nykyään kuitenkin ei-reguloitu ohjelmistokehitys, joka kattaa valtaosan ohjelmistokehityksestä, on adoptoinut ketterät menetelmät (*Agile Adoption Rate Survey Results: February 2008* 2008). Ketterä ohjelmistokehitys pyrkii vastaamaan asiakkaan tarpeisiin mahdollisimman nopeasti saamalla ohjelmiston ulos jakeluun niin nopeasti kuin mahdollista (Beck et al., 2013). Tällä saavutetaan mm. se, että ohjelmisto ei kerkeä liiketoiminnallisessa mielessä vanhentua ennen kuin se päätyy markkinoille. Jotta nopeaan kehityssykliin päästään, tarvitaan kehityksen ympärille prosessi ja au-

tomaatiota, jotta kyvykkyys julkaista uusi versio esimerkiksi päivittäin säilyy. Tämä tarkoittaa, että verifointiprosessin täytyy olla käytännössä lähes automaattinen.

IEC 62304-standardi ei suoraan sano, mitä ohjelmistokehitysmallia tulisi käyttää, vaan asettaa pelkästään vaatimuksia, jotka tulee ottaa huomioon. Se sisältää myös eksplisiittisen toteamuksen ”This standard does not prescribe a specific life cycle model”, joka tarkoittaa, että standardi ei ota laisinkaan kantaa siihen, tulisiko käyttää vesiputousmallista kehitystä vai ketteriä metodologioita (IEC 62304, 2015). Standardi kuitenkin esittelee vaatimukset vesiputoustyylisen V-mallin kautta (IEC 62304, 2015), mikä indikoi vesiputoustyylisen V-mallin olevan vähintään yksi suosituimmista tavoista käyttää standardia. Tutkielman tavoitteena on osoittaa, että yksi ohjelmistokehityksen prosessin iteraatio on mahdollista puristaa ajallisesti niin pieneksi automaation avulla, että ohjelmistokehitys tapahtuu ketterästi. Tämä tapahtuu sallimalla V-mallissa paluu aiempiin kohtiin, eli rikkomalla lineaarinen järjestys ja nopeuttamalla yhden iteraation läpimenoaikaa. Eräässä tutkimuksessa on puolistruktutoidulla haastattelututkimuksella osoitettu, että jo nyt osa lääketieteellisiä laitteita kehittävistä organisaatioista on adaptoinut tämän tyyppisen tavan käytäntöön (McHugh et al., 2013). Tämä antaa lisäosoitusta siitä, että yhdistämällä V-mallinen kehitys ketterän kehityksen kanssa ja automatisoimalla suuren osan iteraatiossa tapahtuvasta toistuvasta työstä, myös ohjelmistopohjaisia lääkinnällisiä laitteita on mahdollista kehittää lyhyellä kehityssyklillä
.

Lainsäädännön kautta tulevat vaatimukset verifioinnista ovat yksi suurimmista tunnistetuista hukkaa aiheuttavista asioista lääketieteellisiä ohjelmistopohjaisia laitteita kehittäessä (McHugh et al., 2013). Tutkimus, jossa ketterän kehityksen toimivuutta lääkinnällisten laitteiden kehityksessä tutkittiin puolistrukturoidulla haastattelulla, toteaa myös, että eniten hyötyä ketteristä menetelmistä saadaan ja on mahdollista hyödyntää nimenomaan V-mallin alapäässä olevissa aktiviteeteissa. Näitä ovat arkkitehtuurin suunnittelu, ohjelmiston suunnittelu ja toteutus, sekä näiden vaiheiden verifiointi.

Tämän luvun fokus onkin nimenomaan tutkia, miten tämän osan syklinopeus ja ketteryys voidaan saada samalle tasolle, kuin reguloimattoman ohjelmiston kehitys automaation avulla. Emme siis keskity tätä enempää ketterien ohjelmistokehitysmenetelmien soveltamiseen ohjelmistopohjaisten lääkinnällisten laitteiden kehitykseen, vaan siirrämme fokuksen siihen, mitä aktiviteetteja jatkuvalla integraatiolla ja tuotantoonviennillä voidaan V-mallin alapäässä automatisoida. Rajaamme myös tässä

tutkielmassa lääkinnällisten laitteiden V-mallin yläpään aktiviteetteja pois, kuten käytettävyystestauksen ja validoinnin, ja keskitymme IEC 62304 rajauksen mukaiseen kehitysprosessiin.

5.2 Luokan 2a laatuvaatimusten kautta tulevien aktiviteettien automaatio

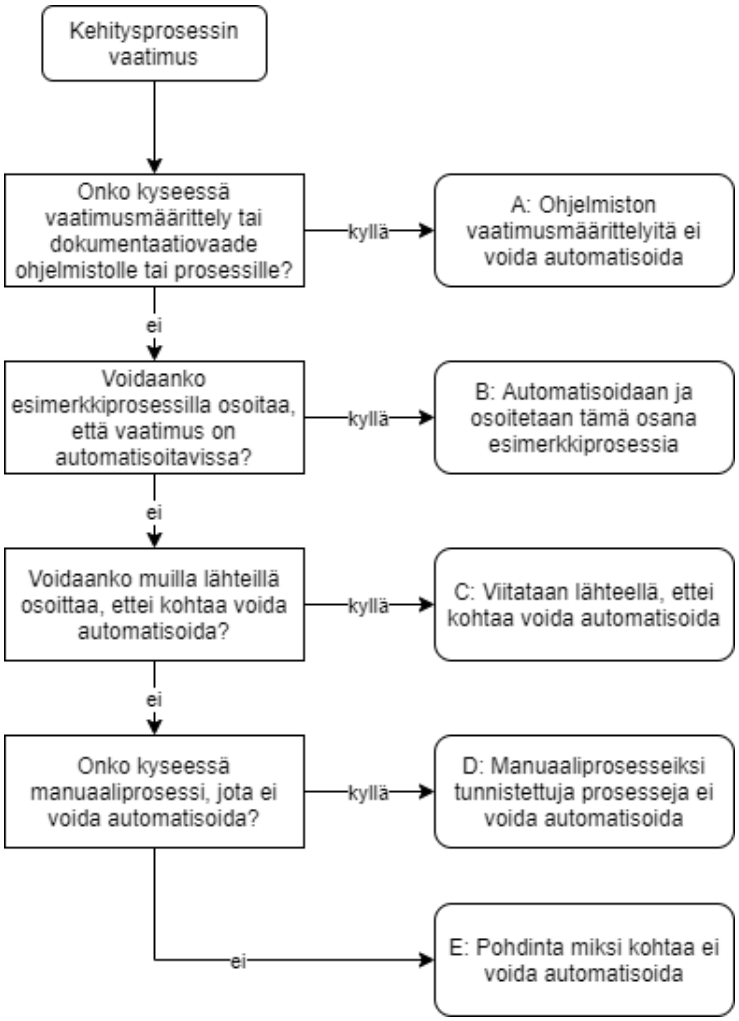
Käymme tässä alaluvussa läpi, mitä laatuvaatimuksia luokan 2a laitteille on, ja perustelemme, miten ne voidaan automatisoida, jos voidaan. Tunnistimme relevantteja vaatimuksia IEC 62304-standardista 26 kpl, olemme numeroineet nämä vaatimukset V-1 - V-26. Alla olevissa laatuvaatimustaulukoissa käymme läpi, mikä vaatimus on, mihin ylempään teemaan se liittyy, minkä tasoisille lääkinnällisille laitteille IEC 62304 standardin mukaan vaatimus kuuluu (A, B, C), mikä lähde vaatimukselle on, miten perustelemme sen automaation toteutuskelpoisuuden alla olevan luokittelun mukaisesti (Kuva 5.1), ja mikä perustelu lopputulokselle on. Olemme lisänneet myös pohdinnan, hyödyttääkö tämä vaatimus ja sen automaatio myös ei-reguloitua ohjelmistokehitystä.

Taulukoiden rakenne

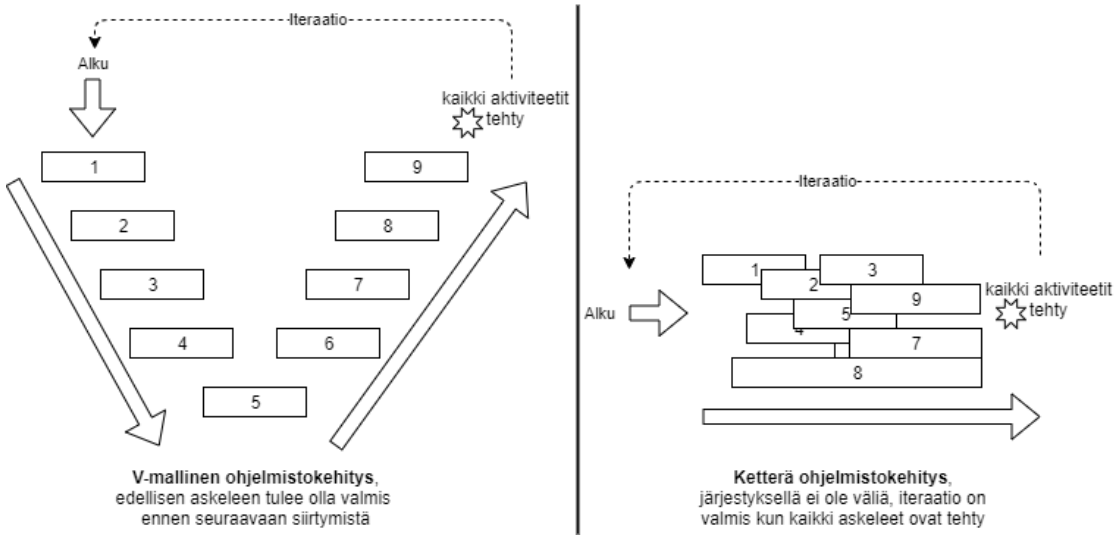
Kentän nimi	**Kuvaus kentästä**
ID	V-1
Vaatimus	Sisältää vaatimuksen, jota kyseisessä taulukossa käsitellään.
Teema	Kuvaus mihin teemaan kyseinen vaatimus liittyy: Dokumentointi, Ohjelmistoprosessi, Ohjelmiston vaatimukset, Ohjelmiston arkkitehtuuri, Ohjelmistosuunnittelu, Ohjelmistokehitys, Integraatio tai Julkaisu.
IEC 62304 luokka	Missä IEC 62304-standarin mukaisissa lääkinnällisissä laitteissa tämä vaatimus tulee huomioida.
Lähde	Kuvaus lähteestä, josta vaatimus on noussut.
Automaation peruste	Kuvan 5.1 mukainen luokitus A-E millä automaatio vaatimukselle voidaan perustella.
Perustelu, voidaanko automatisoida?	Perustelut tämän vaatimuksen automaation mahdollisuuksille.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Pohdinta tämän vaatimuksen hyödyistä myös ei-reguloituun ohjelmistokehitykseen.

Automaation perusteet ovat kirjoitettu vahvasti jatkuvan integraation sekä tuotantoon-

viennin näkökulmasta ja niiden toteutuminen vaatii IEC 62304-standardin tulkinnan, niin ettei kaikkia aktiviteetteja tarvitse suorittaa lineaarisessa järjestyksessä, kuten vesiputous- tai V-mallissa. Automaatio ja sen hyödyt toteutuvat lähtökohtaisesti silloin, kun kaikkia askeleita ei suoriteta lineaarisessa järjestyksessä, vaan varmistutaan ennen tuotantoonvientiä, että kaikki askeleet ovat suoritettu riippumatta niiden suoritusjärjestyksestä. Tätä tulkintaa havainnollistaa Kuva 5.2.



Kuva 5.1: Automaation perusteluiden luokittelu



Kuva 5.2: V-mallin ohjelmistokehitys vs. ketterä ohjelmistokehitys

ID	V-1
Vaatimus	Dokumentoi ohjelmistokehitysprosessi: millaista kehitysmallia käytetään, mitkä ovat lopputulokset prosessista (myös dokumentaatio), kuvaus siitä miten vaatimusmäärittely, testaus ja riskienhallintamenetelmät ovat jäljitettävissä läpi ohjelmistokehityksen kaaren, ohjelmiston konfiguraation ja muutoshallinnan mallit sekä ongelmanratkaisumalli, jolla ohjelmiston eri elinkaaren vaiheissa ongelmia ratkaistaan.
Teema	Dokumentointi
IEC 62304 luokka	A, B, C
Lähde	IEC62304 kohta 5.1.1
Automaation peruste	A
Perustelu, voidaanko automatisoida?	Ei automatisoitavissa, sillä kyseessä on ensisijaisesti kerran tehtävä dokumentaatio, jota tarvittaessa päivitetään myöhemmin, mikäli prosessi muuttuu. Automatisoitu prosessi tulisi kuvata osana tätä kohtaa ja sen kuvauksina voidaan käyttää alla olevia kohtia kunkin prosessin vaiheen automaatiosta.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä osittain, jokainen ohjelmistokehitystiimi käytännössä käyttää jotain ohjelmistokehityksen prosessia. Dokumentaation arvo on laadun säilyttämisessä ja yhdenmukaisuuden saavuttamisessa tiimeissä.

ID	V-2
Vaatimus	Ohjelmistolla tulee olla integraatiotestaussuunnitelma, joka sisältää myös SOUP-komponenttien integraatiotestauksen.
Teema	Dokumentointi
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.1.5
Millä kohdalla automaation toteutuskelpoisuus todetaan	A
Perustelu, voidaanko automatisoida?	Ei, testaussuunnitelman laatiminen automaattisesti ei ole mahdollista, sillä se voidaan rinnastaa osaksi itse ohjelmiston vaatimusmäärittelyä. Päästä-päähän integraatiotestit (end-to-end integration tests) ovat itsessään isompi aihe ja itse testauksen automaatiota on mahdollista rakentaa, johon palaamme tulevissa kohdissa. Integraatiotestaukselle on olemassa paljon tunnettuja tapoja ja niiden mukaan ottaminen itse prosessiin tulee tämän vaatimuksen kautta (Tsai et al., 2001).
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, testaussuunnitelma on oleellinen osa myös ei-reguloidun ohjelmiston kehitystä.

ID	V-3
Vaatimus	Ohjelmistolla tulee olla testaussuunnitelma, joka sisältää mitä lopputuotoksia testataan, mitä verifiointiaktiviteettejä suoritetaan elinkaaren eri vaiheissa, missä kohtaa kehitystä verifiointi tehdään, sekä hyväksymiskriteeristön lopputuloksille.
Teema	Dokumentointi
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.1.6
Millä kohdalla automaation toteutuskelpoisuus todetaan	A
Perustelu, voidaanko automatisoida?	Ei, kyseessä on tiivis osa itse ohjelmiston vaatimusmäärittelyä, jonka automatisointi ei ole mahdollista, sillä se määrittää itse ohjelmistoa ja sen toimintaa. Itse testausautomaatiotakin varten testit on toteutettava manuaalisesti, mutta niiden ajo pystytään automatisoimaan joissakin tapauksissa täysin, palaamme tähän tulevissa kohdissa.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, testaussuunnitelma on oleellinen osa myös ei-reguloidun ohjelmiston kehitystä.

ID	V-4
Vaatimus	Ohjelmistolla tulee olla riskienhallintasuunnitelma, joka käsittelee mitä aktiviteetteja riskien hallintaan tullaan tekemään, sisältäen myös kolmannen osapuolen komponenteista tulevat riskit.
Teema	Dokumentointi
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.1.7
Millä kohdalla automaation toteutuskelpoisuus todetaan	A
Perustelu, voidaanko automatisoida?	Ei, itse riskienhallintausunnitelmaa ei voida automatisoida, mutta osa riskienhallinnan aktiviteeteistä pystytään, kuten esimerkiksi kolmannen osapuolen ohjelmistokomponenttien haavoittuvuustarkastus (engl. Vulnerability check), johon palaamme esimerkkitoteutuksessa luvussa 5.4.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Riskienhallintasuunnitelma ei ole edellytys ohjelmistokehitykselle, mutta riskien huomioiminen kaikessa tekemisessä parantaa laatua. Esimerkiksi EU tasoinen lainsäädäntö GDPR (General Data Protection Regulation) vaatii henkilötietojen käsittelyn osalta riskivaatimuksen, suuri osa ohjelmistoista tekee henkilötietojen käsittelyä, jolloin jonkin tasoinen riskienhallintasuunnitelma on usein osa myös ei-reguloitua ohjelmistokehitystä.

ID	V-5
Vaatimus	Ohjelmistolla tulee olla dokumentaatiosuunnitelma, joka kuvaa mitä dokumentaatiota tulee syntyä ohjelmistokehitysprosessin missäkin vaiheessa sisältäen nimeämiskäytännöt, tarkoitukset sekä kehityksen vastuut katselmoinnille.
Teema	Dokumentointi
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.1.8
Millä kohdalla automaation toteutuskelpoisuus todetaan	A
Perustelu, voidaanko automatisoida?	Ei, dokumentaatiosuunnitelmaa ei voida automatisoida. Osana luvussa 5.4 esiteltävää esimerkkiprosessia automatisoimme kuitenkin osaksi jatkuvan integraation prosessia tavan tarkistaa ohjelmistossa käytettävän nimeämisen sekä koodin laadun (Lint). Se ei kuitenkaan automatisoi vaatimusta ottaa kantaa dokumentaatiossa mm. siihen, miten asioita nimetään.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Dokumentaatiosuunnitelma ei itsessään välttämättä tuota suurta arvoa ohjelmistokehitykseen DevOps ja lean periaatteiden mukaisesti, sen sijaan joitakin yhteisiä käytäntöjä ja dokumentaatioita on tärkeää kuvata kaikissa ohjelmistokehityksen prosesseissa, jotta ohjelmiston jatkokehitys on mahdollista (esimerkiksi arkkitehtuurin kuvaus).

ID	V-6
Vaatimus	Ohjelmistolla tulee olla konfiguraation hallintasuunnitelma.
Teema	Dokumentointi
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.1.9
Millä kohdalla automaation toteutuskelpoisuus todetaan	A
Perustelu, voidaanko automatisoida?	Ei, suunnitelmaa ei voida automatisoida.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Konfiguraationhallintaa tehdään käytännössä jollain tasolla lähes kaikissa ohjelmistokehityshankkeissa, joten jonkin tason hallintasuunnitelma usein syntyy myös ei-reguloituun ohjelmistokehitykseen.

ID	V-7
Vaatimus	Dokumentoi ohjelmistokehitystyökalut ja niiden versiot (mm. kääntäjän versiot ja makefile-tiedostot).
Teema	Dokumentointi
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.1.10
Millä kohdalla automaation toteutuskelpoisuus todetaan	A ja B
Perustelu, voidaanko automatisoida?	Kyllä osittain, ohjelmistokehityksessä käytettävät työkalut voidaan kuvata mm. erilaisissa konfiguraatiotiedostoissa, joita säilytetään osana versiohallintaa. Muutokset näihin työkaluihin kulkevat siis osana kehitysprosessia ja niiden jäljitettävyys ja verifoiminen onnistuu näin normaalin prosessin mukaisesti. On kuitenkin olemassa osia, joiden dokumentaatio automaattisesti ja/tai osana kehitysprosessia ei ole mahdollista. Yhtenä esimerkkinä tällaisesta voidaan pitää esimerkiksi ajoalustaa, jolla sovellus ajetaan.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Ei, lähtökohtaisesti agile-tiimeissä käytettävien ohjelmistokehityskomponenttien dokumentointi ei itsessään tuota lisäarvoa, joka lean periaatteen mukaan on tällöin turhaa.

ID	V-8
Vaatimus	Muutoskohteiden pitää olla versiohallinnassa jo ennen verifiointia
Teema	Ohjelmistoprosessi
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.1.11
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, verifiointi voidaan rakentaa CI putkeen, joka perustuu versiohallinnan päälle. Näin ei ole mahdollista, että verifiointi ajettaisiin ennen muutoskohteiden olemista versiohallinnassa, sillä vain versiohallinnassa oleville kohteille ajetaan verifiointi. Tämä vaatimus täyttyy käytännössä automaattisesti esimerkikkiprosessin mukaisessa sovelluskehityksessä.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, käytännössä jatkuva integraatio vaatii tätä ja sen hyödyt ovat myös ei-reguloidun ohjelmistokehityksen puolella (Meyer, 2014).

ID	V-9
Vaatimus	Ohjelmistolla tulee olla tunnettujen heikkouksien tunnistamisen suunnitelma, joka ottaa valittujen teknologioiden heikkoudet huomioon.
Teema	Dokumentointi/Ohjelmistoprosessi
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.1.12
Millä kohdalla automaation toteutuskelpoisuus todetaan	A ja B
Perustelu, voidaanko automatisoida?	Dokumentaation automatisointi ei ole mahdollista tässäkään kohtaa, mutta valittujen teknologioiden automaattinen heikkouksien tunnistaminen osana automatioitua jatkuvaa integraatiota on mahdollista (Wang et al., 2010). On siis olemassa erilaisia automatisoituja työkaluja näiden heikkouksien tunnistamiseen, mutta tämän lisäksi valituilla teknologioilla on ns. tunnettuja heikkouksia, joiden dokumentaatio tulee tehdä manuaalisesti. Yhtenä esimerkkinä voidaan nostaa yleisesti javascript kehityksessä käytetty npm (Node Package Manager), jossa usealla paketilla on riippuvuuksia hyvin paljon, mikä itsessään luo haavoittuvuuden hallitsemattomuuden kautta (Decan et al., 2017).
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Yleiseen tietoturvalliseen ohjelmistokehitykseen kuuluu heikkouksien tunnistaminen (mm. OWASP top 10), joten näin ollen myös ei-reguloitu ohjelmistokehitys hyötyy heikkouksien systemaattisesta tunnistamisesta.

ID	V-10
Vaatimus	Ohjelmistolla tulee olla kuvaus järjestelmävaatimuksista, joka sisältää funktionaaliset vaatimukset, datamallit, ohjelmiston ja laitteiston väliset vaatimukset, ajonaikaiset hälytykset, turvallisuusvaatimukset, käyttöliittymävaatimukset, tietokantavaatimukset, käytön vaatimukset, verkkovaatimukset ja käyttäjien ylläpitovaatimukset.
Teema	Ohjelmiston vaatimukset
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.2.1 ja 5.2.2
Millä kohdalla automaation toteutuskelpoisuus todetaan	A ja B
Perustelu, voidaanko automatisoida?	Osittain, suuri osa näistä vaatimuksista on automatisoitavissa niin, että itse kuvaus syntyy osana ohjelmistokehitystä ja se voidaan generoida dokumentaatioksi automaattisesti, mikäli näin halutaan. Itse kuvaukset kuitenkin elävät osana prosessia ja versiohallintaa, ja niiden muutokset käyvät saman prosessin läpi kuin kaikki muutkin muutokset sovellukseen. Tulemme esimerkkiprosessissa ja -sovelluksessa kuvaamaan miten mm. suorituskykyvaatimusten yhteenvetodokumentaation generointi automatisoidaan ja miten ohjelmiston sekä laitteiden väliset vaatimukset kuvataan. Tässäkin kohdassa on kuitenki kohtia, joita ei täysin voida automatisoida. Esimerkiksi kuvaus auditointilokista tulee kuvata erikseen, sillä kyse on enemmän ohjelmiston vaatimusmäärittelystä, kuin itse teknisestä toteutuksesta.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Mikäli ei-reguloidussa ohjelmistokehityksessä käytetään myös samoja työkaluja (esim. Docker ja Terraform), syntyvät samat vaatimukset/kuvaukset infrastruktuurille tätä kautta. Näiden vaatimusten generointi esimerkiksi osaksi teknistä tiedostoa ei kuitenkaan ole ei-reguloitua ohjelmistokehitystä hyödyttävää.

ID	V-11
Vaatimus	Ohjelmistolla tulee olla kuvaus vaatimusten verifioinnista.
Teema	Ohjelmiston vaatimukset
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.2.6
Millä kohdalla automaation toteutuskelpoisuus todetaan	A
Perustelu, voidaanko automatisoida?	Ei, tällä vaatimuksella tarkoitetaan dokumentaation verifiointia, eli että dokumentaatio ei ole ristiriidassa itsensä kanssa, riskienhallinta on otettu huomioon jne.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Mikäli ohjelmistolla on järkevä vaatimustenhallintaprosessi, tämä hyödyttää ristiriitojen ratkaisua myös ei-reguloidussa ohjelmistokehityksessä.

ID	V-12
Vaatimus	Muunna ohjelmiston vaatimukset arkkitehtuurikuvauksek-si.
Teema	Ohjelmiston arkkitehtuuri
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.3.1
Millä kohdalla automaation toteutuskelpoisuus todetaan	A
Perustelu, voidaanko automatisoida?	Ei, kyseessä on arkkitehtuurikuvauksen rakentaminen ottaen ohjelmiston vaatimukset huomioon. Tämä määrittää itse ohjelmistoa, joten sen automaatio on käytännössä mahdotonta nykyteknologialla.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Arkkitehtuurikuvaus on olennainen osa ohjelmistokehitystä, joten tämä on yleinen osa myös ei-reguloitua ohjelmistokehitystä.

ID	V-13
Vaatimus	Kuvaa rajapintojen arkkitehtuuri ja dokumentoi ne.
Teema	Ohjelmiston arkkitehtuuri
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.3.2
Millä kohdalla automaation toteutuskelpoisuus todetaan	A ja B
Perustelu, voidaanko automatisoida?	Kyllä osittain, osoitamme osana esimerkkiohjelmistoa ja sen jatkuvaa integraatiota, miten ohjelmiston annotaatioista generoidaan openAPI-kuvauksen mukainen rajapintadokumentaatio. Tämän lisäksi arkkitehtuurikuvauksessa tulee ottaa kantaa ylätasolla eri ohjelmistokomponenttien välisestä suhteesta, tämän automaatiota ei suositella, sillä kyseessä on enemmänkin vaatimusmäärittely ohjelmistolle kuin sen lopputuloksena syntyvä dokumentaatio. Tässä esimerkissä esitämme yhden tavan kuvata sekä dokumentoida rajapinnat niin, että se elää jatkuvasti ohjelmiston mukana. Tämä mekanismi on kuitenkin teknologiakohtainen.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, yleisesti rajapinnat kuvataan ja dokumentoidaan, esimerkiksi openAPI kuvaukset ovat laajasti käytössä myös ei-reguloidussa ohjelmistokehityksessä.

ID	V-14
Vaatimus	Kuvaa SOUP-komponenttien vaatimukset.
Teema	Ohjelmiston arkkitehtuuri
IEC 62304 luokka	B, C
Lähde	IEC62304 kohdat 5.3.3 ja 5.3.4
Millä kohdalla automaation toteutuskelpoisuus todetaan	A ja B
Perustelu, voidaanko automatisoida?	Kyllä osittain, itse vaatimuksia ei voida automatisoida, mutta niiden verifiointi pystytään osittain verifioimaan osana jatkuvaa integraatiota. Kävimme tähän liittyvää ongelmatiikkaa sekä sen mahdollisia automaatioratkaisuita läpi alaluvussa 4.6.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Ei, SOUP on relevantti lähinnä reguloidussa ohjelmistokehityksessä.

ID	V-15
Vaatimus	Verifioi ohjelmiston arkkitehtuuri.
Teema	Ohjelmiston arkkitehtuuri
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.3.6
Millä kohdalla automaation toteutuskelpoisuus todetaan	D
Perustelu, voidaanko automatisoida?	Ei, tällä vaatimuksella haetaan ohjelmiston arkkitehtuurin läpikäyntiä ja verifiointia, että se ottaa huomioon mm. ohjelmiston vaatimukset sekä riskit, joita esimerkiksi SOUP-komponenttien käyttöön liittyy.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Jonkin tasoista verifiointia arkkitehtuurista tehdään myös ei-reguloidussa ohjelmistokehityksessä, varsinaista automaatiota tälle ei voida tehdä kummallakaan puolella.

ID	V-16
Vaatimus	Jaa ohjelmisto moduuleihin.
Teema	Ohjelmistosuunnittelu
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.4.1
Millä kohdalla automaation toteutuskelpoisuus todetaan	D
Perustelu, voidaanko automatisoida?	Ei, kyseessä on vaatimus ohjelmistosuunnitteluun, jolla suositetaan ison ohjelmiston jakamista moduuleihin. Tämä on järkevää myös MDR:n näkökulmasta, sillä näin eri moduuleille voidaan määrittää eri lääkinnällisen laitteen luokat ja osa moduuleista voidaan vapauttaa lääkinnällisen laitteen statuksesta, jolloin standardin ja lainsäädännön vaatimukset eivät koske sitä. On kuitenkin järkevää ottaa samat riskienvaatimusmäärittelyt käyttöön näillekin komponenteille, jotta ne ovat yhteensopivampia muiden moduuleiden kanssa (ISO 14971, 2009).
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Ohjelmistokehitystä tehdään yleisesti modulaarisesti esimerkiksi mikropalveluarkkitehtuurissa, joten ajatusmaailmallisesti tämä ei eroa ei-reguloidun ja reguloidun ohjelmistokehityksen välillä. Automaatiota tämän osalta ei saada kumpaankaan.

ID	V-17
Vaatimus	Tee ohjelmistomoduuleiden verifiointi testaamalla.
Teema	Ohjelmistokehitys
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.5.2
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, esittelemme esimerkkiprosessissa tavan tehdä automaatiotestausta osana jatkuvaa integraatiota, jossa jokaista muutosta vastaan ajetaan tarvittavat testit sekä dokumentoidaan niiden lopputulokset. Tällä pystymme automatisoimaan ohjelmistomoduulien verifioinnin käytännössä täysin.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, testauksen automaatiosta on hyötyä myös ei-reguloidussa ohjelmistokehityksessä. Ohjelmistoja myös lähtökohtaisesti testataan ei-reguloidussakin kehityksessä.

ID	V-18
Vaatimus	Määritä vaatimukset ohjelmistomoduulin hyväksynnälle.
Teema	Ohjelmistokehitys
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.5.3 ja 5.5.5
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, tämä vaatimus liittyy vahvasti ylläolevaan vaatimukseen (Tee ohjelmistomoduulien verifiointi testaamalla). Nämä vaatimukset määritetään osana ohjelmistokehitystä eri tyylisiksi automaatiotestitapauksiksi, jotka verifioidaan osana kehitysprosessia ja voimme näin taata niiden oikeellisuuden.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, sama perustelu kuin vaatimuksessa V-17.

ID	V-19
Vaatimus	Yhdistä / paketoi ohjelmistokomponentit -moduuleiksi, sekä varmista integraation onnistuminen ja aja testit.
Teema	Integraatio
IEC 62304 luokka	B, C
Lähde	IEC62304 kohdat 5.6.1-4
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, osana esimerkkiprosessia toteutamme jatkuvan integraatioprosessin, jossa ohjelmistokomponentit paketoidaan, versioidaan sekä testataan päästä-päähän testeillä (end-to-end testing).
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, sama perustelu kuin vaatimuksessa V-17.

ID	V-20
Vaatimus	Suorita regressiotestaus ohjelmistokomponenttien integraation jälkeen, joka sisältää testitulokset.
Teema	Integraatio
IEC 62304 luokka	B, C
Lähde	IEC62304 kohta 5.6.6
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, osana jatkuvaa integraatioprosessia voidaan ajaa laaja automaatiotestaus paketoituun ohjelmistoon, joka varmistaa sen ettei regressiota ole päässyt syntymään ohjelmistoon. Käymme tämän läpi osana esimerkkiprosessia, joka ajaa nämä testit sekä dokumentoi testitulokset osaksi paketoitua kokonaisuutta.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, sama perustelu kuin vaatimuksessa V-17.

ID	V-21
Vaatimus	Suorita koko ohjelmistojärjestelmän testaus hyväksymiskriteereitä vasten aina muutosten jälkeen. Anomaliat tulee käsitellä virheenkorjausprosessin mukaisesti.
Teema	Integraatio
IEC 62304 luokka	B, C
Lähde	IEC62304 kohdat 5.6.8 ja 5.7.1-3
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä osittain, voimme automatisoida erilaisten virhetilanteiden generoinnin virheenhallintaprosessin mukaisesti, jotka liittyvät alkuperäiseen muutospyyntöön, jonka verifioinnin yhteydessä anomalia on noussut esille.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, sama perustelu kuin vaatimuksessa V-17.

ID	V-22
Vaatimus	Varmista että verifiointi on suoritettu.
Teema	Julkaisu
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.8.1
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, voimme rakentaa prosessin niin, että tuotantoonvienti (jatkuva tuotantoonvienti) on mahdollista vain mikäli jatkuva integraatioprosessi on mennyt läpi täysin ilman ongelmia ja kaikki katselmoinnit sekä verifioinnit on suoritettu. Näin voimme taata, että tuotantoon / jakeluun ei päädy ohjelmistopaketteja, joita ei ole verifioitu.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, mikäli verifiointi on osa prosessia. Tällöin on tärkeä varmistua, että verifiointi on suoritettu.

ID	V-23
Vaatimus	Dokumentoi ja arvio tiedetyt poikkeamat.
Teema	Julkaisu
IEC 62304 luokka	Kaikki, B,C
Lähde	IEC62304 kohdat 5.8.2 ja 5.8.3
Millä kohdalla automaation toteutuskelpoisuus todetaan	D
Perustelu, voidaanko automatisoida?	Ei, mikäli kehitys- tai verifiointivaiheessa tulee esille, että sovelluksessa on jokin poikkeama (mm. SOUP:hin liittyvä), tulee se dokumentoida osaksi ohjelmistoa. Tämän automatisointi ei ole mahdollista, sillä se vaatii tuoteomistajan hyväksynnän, joka puolestaan on manuaaliprosessi, eikä sitä saa automatisoida.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Tiedetyt poikkeamat on hyvä dokumentoida myös ei-reguloidussa ohjelmistokehityksessä.

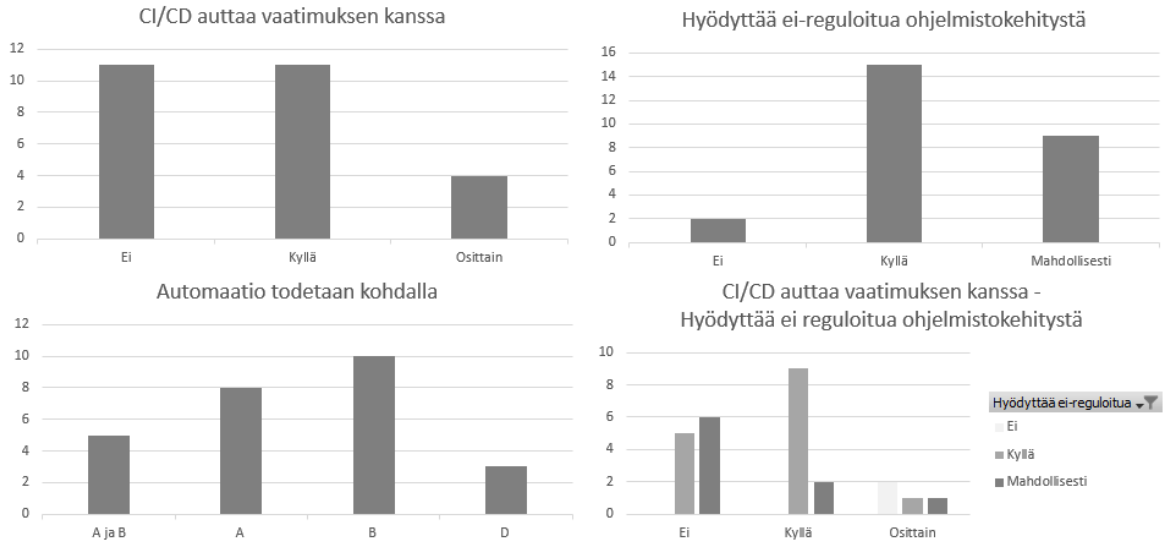
ID	V-24
Vaatimus	Dokumentoi versio.
Teema	Julkaisu
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.8.4
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, jatkuva integraatio tuottaa versionumerot, sekä yhdistää kaiken relevantin dokumentaation siihen, jotta jäljitettävyys säilyy läpi muutoksen. Kaikki dokumentaatio yhdistetään kyseiseen versioon, jota esimerkkiprosessissa pidetään versiohallinnan yhteydessä.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä mikäli versiointia tehdään, automaatio sopii sellaisenaan myös ei-reguloituun ohjelmistokehitykseen.

ID	V-25
Vaatimus	Arkistoi ohjelmisto, sisältäen dokumentaation.
Teema	Julkaisu
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.8.7
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, jatkuva tuontatoonvientiprosessi voi tallentaa lopputuloksena syntyneen artifaktin sekä kaiken siihen liittyvän dokumentaation arkistoon, jolloin kaikkiin aiempiin ohjelmistoversioihin sekä dokumentaation on mahdollista palata milloin vain.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, mikäli ohjelmiston artifakteja arkistoidaan (näin tehdään usein, jos jatkuva tuontantoonvienti on käytössä).

ID	V-26
Vaatimus	Varmista, että tuotantoon viety ohjelmisto ei ole korruptoitunut.
Teema	Julkaisu
IEC 62304 luokka	Kaikki
Lähde	IEC62304 kohta 5.8.8
Millä kohdalla automaation toteutuskelpoisuus todetaan	B
Perustelu, voidaanko automatisoida?	Kyllä, jatkuvan tuotantoonvientiprosessin tulee lopuksi varmistaa, että tuotantoon viety versio on identtinen sen kanssa, mitä vasten verifioinnit on tehty. Yksi mahdollisuus toteuttaa tämä on tiivisteen laskeminen ohjelmistokokonaisuudesta ja näiden vertaaminen tuotantoonviennin yhteydessä. Käymme tämänkaltaisen implementaation läpi osana esimerkkiprosessia.
Hyödyttää ei-reguloidun ohjelmiston kehitystä	Kyllä, myös ei-reguloidun ohjelmiston tuotannon tulee olla korruptoitumaton.

ID	CI/CD auttaa tämän vaatimuksen kanssa	Hyödyttää ei-reguloiduitua ohjelmistokehitystä	Automaatio todetaan kohdalla
V-1	Ei	Mahdollisesti	A
V-2	Ei	Kyllä	A
V-3	Ei	Kyllä	A
V-4	Ei	Mahdollisesti	A
V-5	Ei	Mahdollisesti	A
V-6	Ei	Mahdollisesti	A
V-7	Osittain	Ei	A ja B
V-8	Kyllä	Kyllä	B
V-9	Osittain	Kyllä	A ja B
V-10	Osittain	Mahdollisesti	A ja B
V-11	Ei	Kyllä	A
V-12	Ei	Kyllä	A
V-13	Osittain	Kyllä	A ja B
V-14	Osittain	Ei	A ja B
V-15	Ei	Mahdollisesti	D
V-16	Ei	Kyllä	D
V-17	Kyllä	Kyllä	B
V-18	Kyllä	Kyllä	B
V-19	Kyllä	Kyllä	B
V-20	Kyllä	Kyllä	B
V-21	Kyllä	Kyllä	B
V-22	Kyllä	Kyllä	B
V-23	Ei	Mahdollisesti	D
V-24	Kyllä	Mahdollisesti	B
V-25	Kyllä	Mahdollisesti	B
V-26	Kyllä	Kyllä	B

Taulukko 5.1: Standardin vaatimukset ja niiden automaatio



Kuva 5.3: Standardin vaatimukset ja niiden automaatio

Kuten voimme taulukosta 5.1 ja kuvasta 5.3 nähdä, CI/CD-automaatio voi auttaa

merkittävää osaa tunnistetuista vaatimuksista. Tunnistimme samalla, että ei-reguloitu ohjelmistokehitys hyötyy suuresta osasta vaatimuksia ja todennäköisesti on adoptoinutkin kyseisiä vaatimuksia usein käyttöönsä. Kun reguloidun ohjelmistokehityksen automaatiota verrataan potentiaalisiin hyötyihin reguloimattomassa ohjelmistokehityksessä, voidaan huomata, että erityisesti automatisoitavissa olevat kohdat ovat yhtenäisiä – käytännössä kaikki tunnistetut vaatimukset, jotka olivat automatisoitavissa hyödyttävät tai mahdollisesti hyödyttävät myös reguloimatonta ohjelmistokehitystä.

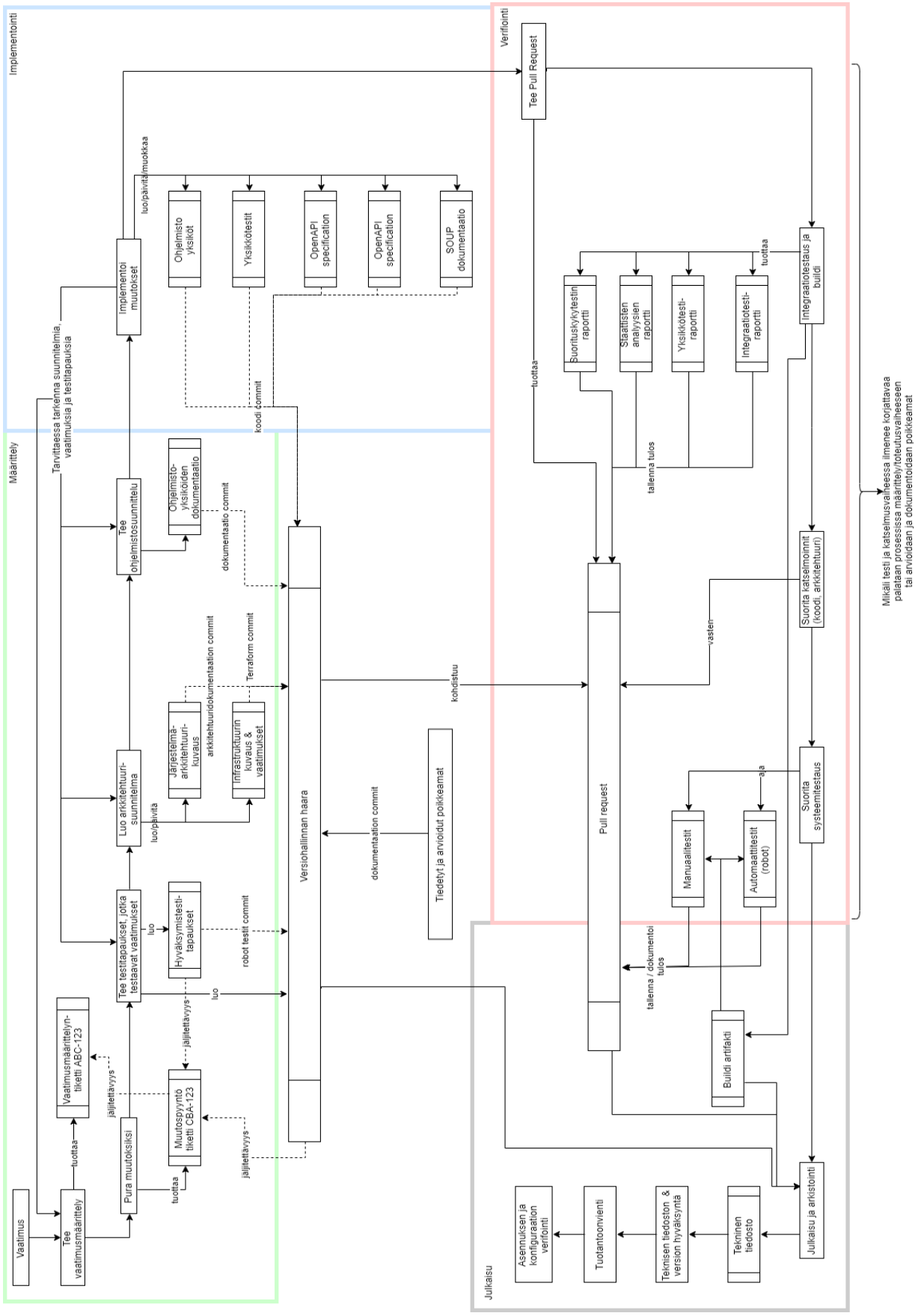
5.3 Luokan 2a ohjelmistopohjaisen lääkinnällisen laitteen kehitysprosesin esimerkki

Esittelemme tässä alaluvussa esimerkinomaisesti ohjelmistopohjaisen lääkinnällisen laitteen mahdollisen kehitysprosessin, jossa on yhdistetty V-mallin, iteratiivisen V-mallin ja ketterän kehityksen piirteitä. Esimerkkiprosessi on rakennettu ohjelmistokehityksen näkökulmasta, eikä näin ollen sisällä kaikki aktiviteetteja, joita itse ohjelmistokehityksen ulkopuolelle lääkinnälliselle laitteelle mahdollisesti kuuluu (esimerkiksi käytettävyystestaus, riskiarvioinnit, validointi, kokonaisarkkitehtuurisuunnittelu jne).

Prosessi on rakennettu pitäen jäljitettävyyttä silmällä, sillä yksi lääkinnällisen laitteen tärkeimpiä asioita on jäljitettävyyden säilyttäminen läpi muutosten aina ensimmäisestä ilmaantumishetkestä laitteen elinkaaren loppuun asti. Käsittelemme tässä alaluvussa myös jäljitettävyyden integriteetin läpi esimerkkiprosessin.

Esimerkkiprosessi koostuu neljästä päävaiheesta: määrittelystä, toteutuksesta, verifioinnista ja julkaisusta. Jokainen päävaihe jakaantuu aktiviteetteihin, jotka on kuvattu prosessissa. Käsittelemme tässä nyt jokaisen päävaiheen erikseen, jotta tarvittava informaatio mm. jäljitettävyydestä saadaan perusteltua.

Prosessi näyttää päällisin puolin hyvin lineaariselta, mutta siinä on mahdollistettu palaaminen aiempiin askeliin tai prosessin vaiheisiin aina tarvittaessa, näin ollen kyseessä ei ole lineaarinen kehitysmalli kuten vesiputousmalli tai V-malli (Balaji ja Murugaiyan, 2012).



Kuva 5.4: Esimerkkiprosessi

5.3.1 Määrittely

Ennen määrittelyvaihetta vaatimus on syntynyt ja sen lähteitä voi olla useita, esimerkiksi asiakastarve. Jokainen vaatimus tuottaa vähintään yhden vaatimusmäärittelyn, joka esimerkkiprosessissa tallennetaan tikettiin. Tällöin vaatimuksen tuottamat määrittelyt saavat uniikin tunnisteen, joka on jäljitettävissä.

Vaatimusmäärittelyn jälkeen vaatimukset puretaan muutospyynnöiksi/tehtäviksi, jokainen muutos dokumentoidaan tiketiksi, joka linkitetään vaatimukseen. Näin linkki vaatimuksen ja tehtävän välillä säilyy. Tämän jälkeen tehtäviä aletaan kuljettaa läpi prosessin – jokaista tehtävää vasten luodaan versiohallinnan haara, joka nimetään muutospyynnön ID-numeron mukaisesti (esim. CBA-123-change-info-text). Tähän versiohallinnan haaraan kerätään kaikki muutokset, mitä prosessin määrittely- ja toteutusvaihessa syntyy, kuten hyväksymistestitapaukset, ohjelmistoarkkitehtuurisuunnitelmat, ohjelmistosuunnittelut ja dokumentaatio.

Määrittelyn vaiheet on prosessissa piirretty lineaarisiksi, mutta niitä voidaan suorittaa epälineaarisessa järjestyksessä. Äärimmäisen tärkeää on kuitenkin, että kaikki muutokset tallentuvat versiohallinnan haaraan, joka on nimetty muutospyynnön ID:n mukaisesti. Näin kaikki määrittely, dokumenaatio ja verifiointitapaukset ovat yhdistettävissä alkuperäiseen muutospyyntöön ja sitä kautta vaatimukseen.

5.3.2 Toteutus

Toteutusvaihe keskittyy erityisesti muutospyynnön, vaatimuksen, tehdyn ohjelmistoarkkitehtuuri- ja ohjelmistosuunnittelun mukaisesti toteuttamaan muutoksen järjestelmään.

Tässä vaiheessa jokainen ohjelmistoyksikkö toteutetaan, sekä varmistetaan, että ohjelmistoyksikölle on olemassa tarvittavat testit verifioimaan toteutus. Mikäli ohjelmistoyksikön toteutus vaikuttaa rajapintakuvauksiin, SOUP-dokumentaatioon tai muuhun dokumentaatioon tehdään tarvittavat muutokset näihin kohtiin, jotta varmistetaan ohjelmiston ja dokumentaation eheys. Näin ollen tässä yhteydessä toteutetaan myös tarvittavat testit käyttöönotettaville SOUP-komponenteille.

Jäljitettävyys on taattu versiohallinnan haaran kautta, jokainen muutoksen on kohdistuttava tiettyyn haaraan, joka liittyy tiettyyn muutospyyntöön. Yhtään muutosta ei voi näin ollen tapahtua ilman, että se linkittyisi johonkin tiettyyn muutospyyntöön

tai vaatimukseen sitä kautta.

5.3.3 Verifiointi

Kaksi edellistä päävaihetta, määrittely ja toteutus, olivat pitkälti dokumentointia ja toteutusta, jossa tämän työn tarkoittamaa automaatiota ei erityisemmin pystytty vielä hyödyntämään tai tätä kautta kehitysprosessia nopeutettua ei-reguloidun ohjelmisto kehitykseen verrattuna. Verifiointivaiheessa kuitenkin automaatio tuottaa jo selkeitä hyötyjä, joilla ohjelmistopohjaisen lääkinnällisen laitteen kehitystä saadaan ketteröitettyä lähelle ei-reguloidun ohjelmisto kehitystä.

Verifiointivaihe alkaa liitospyynnön luonnilla (engl. Pull Request), joka tarkoittaa että versiohallinnan haara, johon kaikki määrittely- ja toteutuspäävaiheen muutokset on kerätty yhteen, pyydetään liitettävän ohjelmiston päähaaraan, eli esimerkiksi ottamaan mukaan tuotannon seuraavaan versioon.

Jatkuvan integraation automaatio huomaa välittömästi tämän pyynnön ja aloittaa automaation suorituksen, tämä alkaa ohjelmiston rakentamisella (eng. Build), joka tuottaa lopputuloksena aiempien versioiden ja nyt pyydettyjen muutosten mukaisen ohjelmiston. Tätä ohjelmistoversioita vasten ajetaan yksikkötestit, integraatiotestit, staattiset analyysit sekä suorituskykytestit. Tässä yhteydessä suoritetaan käytännössä myös alaluvussa 5.4 kuvatut apuskriptit, jotka mm. voivat tuottaa dokumentaatiota yms.

Mikäli automaatio huomaa, että kaikki testit tai muut integraation aktiviteetit eivät onnistuneet automaattisesti, palautetaan tästä tieto kehittäjälle, jolloin palataan määrittely- tai toteutuspäävaiheeseen. Tämän mahdollistaa iteraatio, jota voidaan suorittaa DevOps periaatteen mukaisesti.

Jatkuvan integraation seuraaviin vaiheisiin kuuluu vielä kokonaissysteemin testaus koodi ja arkkitehtuurikatselmoinnin jälkeen. Kaikki nämä tulokset tallennetaan osaksi versiohallinnan haaraa sekä liitospyynnön dokumentaatiota. Näin ollen kaikki muutokset ovat jäljitettävissä aina muutospyyntöön ja sitä kautta vaatimukseen.

Jatkuvan integraation prosessi voi vielä varmistaa versiohallinnan integriteetin, eli tehdä tarkistuksen, ettei yksikään muutos ole tullut versiohallintaan ilman muutoshallinnan tikettinumeroa tai tunnistetta.

5.3.4 Julkaisu

Kun verifiointivaihe on onnistuneesti mennyt läpi, tiedämme että ohjelmisto täyttää ne tekniset vaatimukset mitä sille on asetettu (huom! tämä ei tarkoita, että ohjelmisto olisi hyväksytty lääkinnällinen laite, tähän prosessiin liittyy vielä tämän tukielman ja esimerkin ulkopuolisia aktiviteettja, tässä otetaan kantaa lähinnä IEC 62304-standardin mukaisiin aktiviteetteihin).

Julkaisuvaihe alkaa rakennetun ohjelmiston artifaktin arkistoinnilla. Jatkuvan integraation tuottama artifakti siis tallennetaan arkistointikäyttöä varten pysyvästi. Tämän jälkeen generoidaan tekninen tiedosto lääkinnällisestä laitteesta, todellisuudessa tähän vaiheeseen kuuluu myös muiden osa-alueiden dokumentaation generointeja ja useita alavaiheita, mutta emme tässä esimerkissä tai tutkielmassa ota niihin kantaa. Tämän jälkeen tekninen tiedosto hyväksytään, jonka yhteydessä jatkuva tuotantoonvienti ja sen automaatio voi viedä ohjelmistopohjaisen lääkinnällisen laitteen tuotantoon. Jatkuvan tuotantoonviennin viimeinen tehtävä on verifioida tuotantoon tehty asennus, että se on toteutunut kuten tarkoitettu, esimerkiksi tämä voi tarkoittaa että artifakti on säilynyt eheänä, tai jollain infrastuktuurinkuvauskielellä toteutettu kuvaus on implementoitunut infrastruktuuriksi oikein.

Todellisuudessa julkaisuja tehtäisiin useaan otteeseen eri ympästöihin, ja todennäköisesti myös julkaisu testiympäristöön tehtäisiin esim. käytettävyystestausta varten jo ennen teknisen tiedoston luontia tai hyväksyntää. Tämä käytännössä tarkoittaa, että verifioinnin jälkeen tulee aktiviteetteja, joita on suoritettava ennen julkaisua - tämä on yksi tekijöistä, joka estää reguloidun ohjelmistokehityksen automaattisen julkaisun suoraan onnistuneen integraation jälkeen, mikä olisi mahdollista ei-reguloidussa ohjelmistokehityksessä.

Kaikki artifaktiin ja tekniseen tiedostoon päätyvät muutokset ovat jäljitettävissä versiohallinnan haaran yhdistymishetken (engl. Merge Commit) kautta muutospyyntöön ja siitä edelleen vaatimukseen. Näin ollen pystymme identifioimaan aina tuotantoon asti jokaisen muutoksen, joka prosessin läpi on mennyt.

5.4 Esimerkkitoteutus

Tässä alaluvussa esittelemme alaluvun 5.3 kuvaaman prosessin käytännön toteutuksen. Paneudumme tässä alaluvussa vielä tarkemmin siihen, minkä edun jatkuva integraatio

ja tuotantoonvienti tuovat esimerkkiprosessiin.

Valitsimme esimerkkitoteutukseen yleisesti käytössä olevat työkalut, joita on käytetty myös aiemmin reguloidun ohjelmiston kehitykseen eri kontekstissa (Filion et al., 2017).

Esimerkkitoteutukseen valitut työkalut ja niiden vastuut ovat:

- Atlassian JIRA
  - Muutostenhallinta
  - Julkaisujenhallinta
  - Projektinhallinta
- Atlassian Bitbucket
  - Versionhallinta, pohjalla Bitbucket käyttää Git-versiohallintaa
  - Jatkuvan integraation automaatio (CI-putki)
  - Jatkuvan tuotantoonviennin automaatio (CD-putki)
  - Liitospyyntöjen hallinta (eng. Pull Request management)

Emme käy tässä työssä läpi sen luonteen vuoksi näiden ohjelmistojen asennusta tai niiden välistä integraatiota, vaan keskitymme prosessin ja automaation rakentamiseen näiden tuotteiden päälle.

5.4.1 Muutosten ja vaatimusten hallinta

IEC 62304-standardi sekä aiemmassa luvussa kuvattu esimerkkiprosessi antavat meille hyvän pohjan prosessille, jonka olemme käytännössä implementoineet JIRA-tuotteeseen. Esimerkkiprosessissa toteutimme Kanban metodiikalla taulun, jossa jokaisella muutospyynnöllä on oma tiketti (engl. Ticket), jonka etenemistä seurataan taululla vasemmalta oikealle. Suurimmat motivaatiot Kanban-metodin käyttöön tässä työssä ovat samat kun kanbanin käytössä yleisesti – yksinkertaistaa prosessia ja tuoda työ täysin näkyväksi. Kanban on yleisesti käytössä vastaavan tyylisissä ohjelmistokehitysprosesseissa (Ahmad et al., 2013).

IEC 62304-standardin, sekä kuvan 5.4 mukaisesti tarvitsemme kanban-tauluun tilat (IEC 62304, 2015):

- Backlog, yleinen lista johon kaikki muutospyynnöt nostetaan ennenkuin niiden suunnittelu alkaa

- Määrittely
- Toteutus
- Verifiointi
- Valmis julkaistavaksi

Toteutus-kohdassa olemme yhdistäneet IEC 62304-standardin kohdat 5.3-5.5 ketterän kehityksen mukaisesti (kattaa käytännössä vaatimukset V-12 - V-18), sillä nämä kaikki askeleet voidaan toteuttaa yhdessä, tärkeintä on varmistua, että kaikki askeleet ovat toteutettu ennenkuin muutospyyntö julkaistaan (IEC 62304, 2015). Esimerkkitoteutuksessa huolehdimme varmistamisesta kohdassa *Verifiointi*.

Muutoshallintajärjestelmät, kuten JIRA eivät vakiona sisällä kaikkia lääkinnällisten laitteiden kehitykseen tarvittavia tietokenttiä dokumentointia ja jäljitettävyyttä varten, tästä syystä on tärkeää lisätä kaikki tarvittavat kentät niin, että jokaiselle muutospyynnölle on olemassa seuraavat tietokentät:

- Muutoksen yksilöivä tunniste, käytämme tästä jatkossa termiä tikettinumero
- Vaatimusmäärittely
- Muutoksen otsikko
- Muutoksen kuvaus (tässä työssä ei keskitytä tähän)
- Muutoksen perustelu (tässä työssä ei keskitytä tähän)
- Versionumero, jossa muutos on julkaistu (voidaan täyttää joko etukäteen, eli suunnitella versio, tai jälkikäteen kun muutos liitetään johonkin versioon julkaisua varten)
- Kliininen arviointi (tässä työssä ei keskitytä tähän)
- Riskien arviointi (tässä työssä ei keskitytä tähän)
- Toteutuksesta vastaava henkilö
- Suunnittelusta vastaava henkilö
- Muutoksen tyyppi
  - Ominaisuus
  - Ylläpito
  - Virheenkorjaus (eng. Bug fix)
- Muutoksen työestimaatti (tässä työssä ei keskitytä tähän)
- Muutoksen hyväksyjä (tässä työssä ei keskitytä tähän, mutta automatisoidaan tämän tiedon täyttö osana CI/CD prosessia)
- Muutoksen ilmoittaja / kirjaaja

- Testausmekanismi ja -ohje

Olemme nyt määrittäneet tilat joissa tiketit voivat olla, sekä tietokentät mitä tiketteihin tulee olla kirjattuna. Käytännön elämän helpottamiseksi ja virheiden minimoinniksi JIRA implementaatiossa tulisi vielä määrittää transitiot, joita tiketeillä voi olla ja missä kohtaa mihinkin tietokenttään on välttämätöntä ottaa kantaa ennenkuin seuraavaan vaiheeseen voi siirtyä. Emme kuitenkaan tässä esimerkkitoteutuksessa käy läpi näitä kohtia, sillä ne eivät ole relevantteja itse aiheelle.

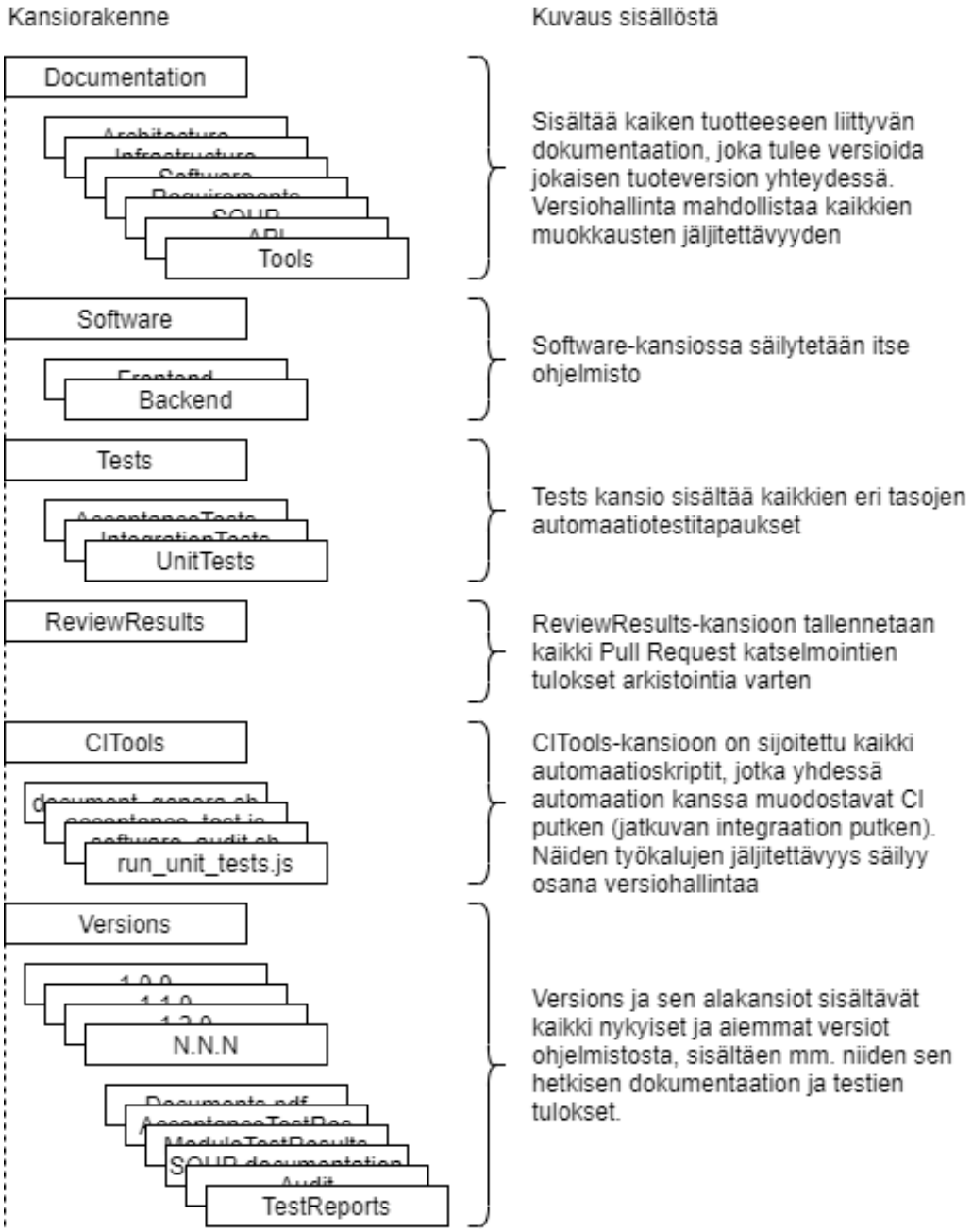
Yllä olevassa esimerkkikonfiguraatiossa vaatimusmäärittelyt, sekä testaustapaukset (engl. Test Cases) ovat laitettu samaan tikettiin itse muutoksen kanssa, hyvin pienissä tuotteissa tämä voi olla järkevää, mutta heti kun tuote alkaa kasvaa, on vaatimusmäärittelyt ja testitapaukset syytä erottaa omiksi tiketeiksi ja kuvata niiden suhde itse muutostiketteihin (IEC 62304, 2015). Tämä järjestely helpottaa jokaisessa versiossa tehtävää testausta, sillä jokainen testitapaus tulee suorittaa jokaisen version yhteydessä, jos kaikki testitapaukset on kuvattu vain osana muutoksia voi kaikkien testitapausten hahmottaminen ja suorittaminen olla haasteellista.

5.4.2 Versiohallinta osana prosessia ja sovelluksen rakenne

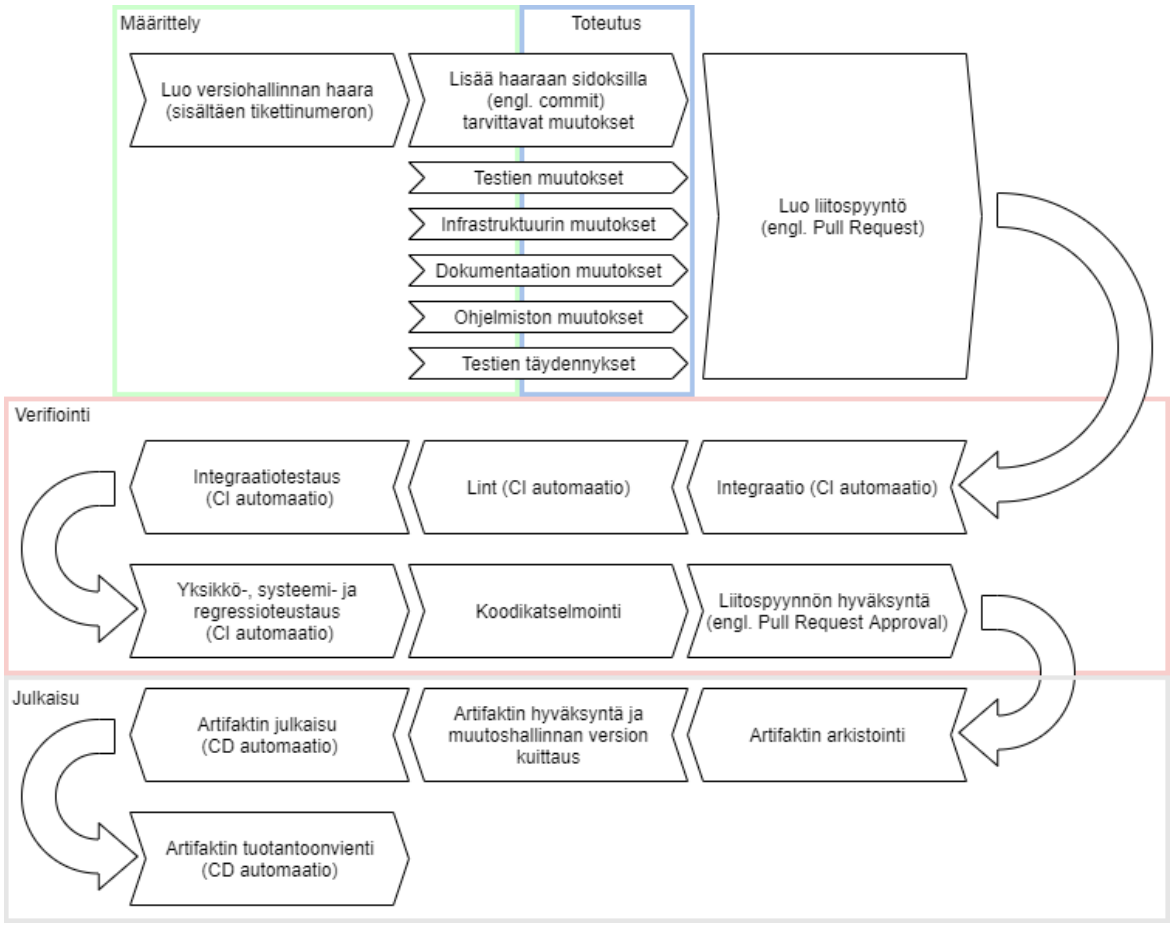
Kun muutostenhallintaan on luotu tiketti edellä kuvatun tiedon ja prosessin mukaisesti voimme siirtyä itse teknisen määrittelyn ja toteutuksen vaiheeseen prosessissa, jota tukee versiohallinta sekä sen sisällä oleva kansiorakenne.

Kuva 5.5 havainnollistaa versiohallinnan kansiorakenteen, joka avaa struktuuria jossa ohjelmistomme ja sen tarvitsemat tukirakenteet säilytetään. Jäljitettävyyden ollessa tärkeää lääkinnällisen laitteen jokaisessa kehitysvaiheessa, tarjoaa versiohallinta (kuten Bitbucket/Git) meille loistavan pohjatyökalun jäljitettävyyden ylläpitoon. Käytännössä jokainen muutos (eng. Commit) versiohallintaan on jäljitettävissä muutosviestissä (eng. Commit message) olevan tikettinumeron kautta varsinaiseen muutospyyntöön – eli JIRA:ssa olevaan tikettiin. Versiohallinta pitää huolen siitä, että jokainen muutos tiedostoihin, olivatpa ne sitten dokumentaatiota, ohjelmistoa tai jatkuvan integraation työkaluja, tulee jäljitettyä ja sillä on yhteneväinen muutospyyntö olemassa.

Tulevissa alaluvuissa käymme tarkemmin mm. läpi mitä työkaluja CITools sisältää, jotta automaation taso ohjelmistopohjaisen lääkinnällisen laitteen kehityksessä voidaan saada tarvittavalle tasolle.



Kuva 5.5: Kansiorakenne versiohallinnassa



Kuva 5.6: Versiohallinan prosessi ja sen kytkeytyminen kokonaisprosessiin

JIRA voidaan ajatella olemaan kattojärjestelmä kaikille muutoksille, joita lääkinnälliseen laitteeseen tehdään. Bitbucket on taas järjestelmä, jossa hallitaan kaikki tekninen dokumentaatio, itse ohjelmiston kehitys ja kaikki työkalut, joita tarvitaan kehityksen onnistumisessa, tämän roolin kokonaisuutta avataan kuvassa 5.6.

5.4.3 Jatkuvan integraation prosessi

Kaksi aiempaa alakohtaa ovat kuvanneet vaatimusten, tehtävien ja versiohallinnan prosessit, tässä kappaleessa kuvaamme millaisen CI-automaation olemme toteuttaneet esimerkkiä varten Atlassian Bitbucket Pipelines päälle. Pipelines on CI/CD-automaation ajoalusta, se tarjoaa valmiita integraatioita useisiin palveluihin sekä mahdollistaa omien CI/CD-komentojen ja ajojen määrittelyn. Se siis orkestroi koko CI- ja CD-putkien toiminnan alusta loppuun. Käytämme tässä esimerkissä Atlassian Bitbucket Pipeline-tuotetta sen helpon integroituvuuden vuoksi Bitbucket- ja JIRA-tuotteisiin

(*Bitbucket Pipelines* 2019). CI/CD-automaation prosessin voisi yhtä hyvin toteuttaa muillakin tuotteilla, eikä standardit tai käytännöt ota kantaa mitä ajoalustaa tulisi käyttää (IEC 62304, 2015).

Kuten kuvan 5.6 prosessista voi huomata olemme automatisoimassa kohtia integraatio (CI), lint (CI), integraatiotestaus (CI), yksikkö-, systeemi, ja regressiotestaus (CI), artifaktin julkaisu (CD) ja tuotantoonvienti (CD). Tavoitetila automaatiolla on poistaa manuaalisia askeleita regulaation vaatimassa ohjelmistokehitysprosessissa. Olemme jo taulukossa 5.1 listanneet aktiviteetit, jotka pystytään automatisoimaan CI prosessissa. Tässä alaluvussa osoitamme nyt, miten automaatio voidaan totetuttaa.

Vaikka Bitbucket Pipelines -tuote sisältää valmiita integraatioita ei se itsessään tarjoa työkaluja, joilla ohjelmistopohjaisen lääkinnällisen laitteen jatkuva integraatio voidaan saavuttaa. Alla olemme kuvanneet työkaluja, joita olemme toteuttaneet teknisenä kontribuutiona, jotta jatkuva integraatio voidaan saavuttaa. Jatkuvan integraation putki (CI-putki) orkestroi näiden työkalujen suoritusta ja varmistaa niiden suorituksen automaattisesti jokaisen liitospyynnön yhteydessä. Mikäli putken jokin osa epäonnistuu tai suorituksen aikana nousee esille virheitä toteutuksessa pitää orekstroija huolen, että tieto palautuu kehittäjälle. Kun kaikki työkalut ovat onnistuneesti suoritettu mahdollistaa integraation putki uuden version generoinnin ja mahdollisen tuotantoonviennin.

Työkalut eri vaatimusten automatisointiin ja kuvaus niiden toimintaperiaatteista:

metadata.json: tiedosto versiohallinnassa sisältää informaation mikä on versionumero, joka on kehityksen alla. Tämä tieto ohjaa CI ja CD prosessia vahvasti mm. dokumentaation versioinnissa.

document_generator.js ja Vaatimukset V-7 ja V-24: muodostamme CI putkessa automaattisesti dokumentaation artifaktia, arkistointia ja teknistä tiedostoa (engl. Technical File) varten Documentation-kansiosta, tähän käytämme md-to-pdf nimistä kirjastoa, jolla käymme läpi dokumentit, keräämme ne yhteen ja muodostamme PDF:n, joka tallennetaan Versions-kansioon metadata.json määräämän versionumeron alle.

Vaatimus V-8: Voimme osoittaa täyttävämme vaatimuksen suoraan sillä, että mikään CI putken osa ei voi suorittua ilman, että ohjelmisto on versiohallinnassa, sillä CI perustuu versiohallinnan päälle. Kaikki muutokset tuodaan näin osaksi versionhallintaan ennen testausta kuten esimerkkiprosessi 5.3 osoittaa.

software_audit.sh ja Vaatimus V-9: NPM teknologian käyttö mahdollistaa tunnet-

tujen haavoittuvuuksien automaattisen listauksen *npm audit* komennolla, tämä on osa vaatimuksen V-9 tarkoittamaa heikkouksien tunnistamiseta. *software_audit.sh* työkalu ajaa tässä kontekstissa mm. npm audit komennot kaikille paketeille ja dokumentoi niiden lopputulokset *Versions/<N.N.N>/Audit/*-kansioon, josta jää jälki sen hetkisistä tunnetuista heikkouksista. Osana jatkuvan integraation putkea ajetaan myös samat komennot ja mikäli tunnettuja heikkuksia löytyy ei ajoa päästetä läpi (ts. ne on korjattava ennen tuotantoonvientiä). Tämän lisäksi tunnetuista heikkouksista, jotka ovat mm. arkkitehtuurivaiheessa syntyneet tulee tehdä dokumentointi, tätä ei voida automatisoida.

softare_requirements_assembler.js ja Vaatimus V-10: IEC 62304 vaatii ohjelmiston vaatimusten kuvaamisen ja dokumentaation. Esimerkkiprosessin mukaisesti ohjelmistosuunnittelu ja sitä kautta vaatimusmäärittelyt dokumentoidaan osaksi itse muutosta, jotta jäljitettävyys muutoksille voidaan taata. Lean henkisesti turhan työn tekemistä tulee välttää, joten ajatusmalli tässä on kuvata samat vaatimukset itse sovellukselle teknisesti kuin teknistä tiedostoa (engl. Techival File) varten. Käytännössä tämä tarkoittaa, että software_requirements_assembler.js generoi mm. Docker- sekä Terraform-tiedostoista osan V-10 vaatimista määrittelyt automaattisesti kansioon *Documentation/Software/*. Kaikkia vaatimuksen V-10 dokumentointitarpeita ei voida automaattisesti kerätä ja näitä ylläpidetään manuaalisesti osana dokumentaatiota.

api_documentation_generator.sh ja Vaatimus V-13: IEC 62304 vaatii, että rajapintojen arkkitehtuuri on olemassa ja rajapinta on dokumentoitu. Koko arkkitehtuurin kuvausta ei voida automatisoida, mutta REST-rajapinnan teknisestä kuvauksesta on mahdollista generoida dokumentaatio automaattisesti. Toteuttamamme skripti api_documentation_generator.sh ajaa openApi koodigeneroijan docker kontissa, joka lukee ohjelmiston rajapintakuvauksen yaml tiedostosta ja generoi siitä ihmislukuisen dokumentaation *Documentation/API/* kansioon automaattisesti.

soup_requirement_handler.sh ja Vaatimus V-14: Vaatimus V-14 vaatii, että SOUP-komponenteille tehdään funktionaaliset- ja performassinvaatimukset. soup_requirement_handler.sh auttaa osittain tässä kokoamalla kaikkien soup-komponenttien dokumentaatiot yhteen *Versions/SOUPDocumentation*-kansion alle, sekä ajamalla SOUPTestit, joilla funktionaalisia- ja performassinvaatimuksia voidaan automatisoida. Näiden testien tulokset talletetaan myös *Versions/<N.N.N>/SOUPDocumentation* kansion alle.

run_unit_tests.js ja Vaatimukset V-17 ja V-18: Vaatimukset V-17 ja V-18 kuvaavat tarvetta tehdä verifiointi ohjelmistomoduuleille, jotka voidaan testiautomaatiolla kattaa. Osana kehitysprosessia tulee määrittää vaatimukset moduulien hyväksynnälle ja kääntää nämä vaatimukset testiautomaatiotapauksiksi. run_unit_tests.js ajaa testiautomaatiot ja tuottaa niistä syntyneen raportin *Versions/<N.N.N>/ModuleTestResults/<Moduulin nimi>/* -kansion alle.

acceptance_test.sh ja Vaatimukset V-19, V-20 V-21: osana CI putken loppua ajamme kaikki hyväksymistestit, jotka on kirjoitettu kehitysvaiheessa ja varmistamme näiden läpimenon. Hyväksymistestien raportit tallennetaan *Versions*-kansion versionumeron alle AcceptanceTestResults kansioon.

Vaatimus V-22: Jatkuvan tuotannon pipeline on rakennettu niin, ettei tuotantoonvientiä voi käynnistää, mikäli jatkuvan integraation (CI) vaihe ei ole mennyt kokonaisuudessaan läpi. Näin ollen automaatio varmistaa, että verifiointi on suoritettu ennenkuin tuotantoonvienti voidaan aloittaa.

Vaatimus V-25: Osana artifaktin generointia CI prosessissa arkistoimme artifaktin, jolloin vaatimus arkistoinnista, joka sisältää dokumentaation tulee täytettyä.

Vaatimus V-26: Osana tuotantoonvientiprosessia varmistetaan, että artifakti joka on viety tuotantoon on sama mikä paketoitiin osana jatkuvaa integraatiota. Tämä tehdään ajamalla sha1checksum artifaktipaketille arkistoinnin yhteydessä ja tuotantoonsiirron jälkeen.

Arkistointi ja versiointi: Tekninen toteutus arkistoinnista toimii esimerkkisovelluksessa kumuloimalla versiohallinnan *Versions* alle kaikki versiot ohjelmistosta CITools-skriptien avulla. Tämä johtaa tuplatietoon, sillä samat tiedot ovat versiohallinnassa muissa kansioissa – versiohallinnalla ei kuitenkaan ole tietoa julkaistuista versioista ja IEC 62304 vaatii kaikkien julkaistujen versioiden ja niiden dokumentaation säilytyksen. Käytännössä CITools-skriptit siis kopioivat aina julkaistavaa versiota varten kaiken dokumentaation ja muut tarvittavat osat julkaisua varten *Versions*-kansion alle.

6 Johtopäätökset

Olemme tässä tutkielmassa käsitelleet ohjelmistopohjaisten lääkinnällisten laitteiden jatkuvaa integraatiota ja toimittamista reguloidussa ohjelmistotuotannossa. Motivaatio tämän tutkielman kirjoittamiseen on ollut EU:n uusi lainsäädäntö, joka tiukentaa erityisesti ohjelmistopohjaisten lääkinnällisten laitteiden kehitystä. EU:n Medical Device Regulation (MDR) on tullut voimaan vuonna 2017, mutta sen siirtymäaika on vuoden 2020 toukokuun loppupuolelle. Moni ohjelmistopohjaisia lääkinnällisiä laitteita kehittävä organisaatio pyrkii ketteröittämään omaa lääkinnällisen laitteen kehitysprosessia pitääkseen kehityksen relevanttina markkinoilla.

Tässä tutkielmassa olemme esimerkin kautta käyneet läpi ohjelmistopohjaisen lääkinnällisen laitteen kehitystä, sekä tunnistaneet vaatimuksia mm. IEC 62304-standardista, jotka vaikuttavat kehitykseen, sekä pyrkineet osoittamaan näille automaation mahdollisuuksia.

Tutkielman yhtenä lähtökohtana oli tutkia, mitkä ovat lääkinnällisten laitteiden EU:n mukaisen luokittelun luokan 1 ja 2a erot (RQ1). Luokitteluero tulee siitä, millaista riskiä potilaalle laite voi tuottaa. Riski on käsite, joka on ilmenemistodennäköisyyden ja vahingon vakavuuden yhdistelmä. Lainsäädäntö mm. MDR:n aikana kuvaa esimerkiksi ohjelmistopohjaisille lääkinnällisille laitteille suhteellisen tarkastikin sen, mihin luokkaan niiden tulee kuulua riskinsä perusteella. Regulatooriset vaatimukset luokan 1 ja 2a välillä eroavat merkittävästikin; luokan 1 lääkinnällisille laitteille riittää itseilmoitus, kun taas luokan 2a laitteet tulee auditoida ilmoitetun laitoksen toimesta. Lisääntynyt regulaatio hidastaa ohjelmistokehitystä. Näin ollen siirtymä luokasta 1 luokkaan 2a ei ole täysin triviaalia tai tapahdu automaattisesti.

Miksi ohjelmistopohjaiset lääkinnälliset laitteet ovat EU:n uuden lainsäädännön mukaisesti lähtökohtaisesti luokassa 2a eivätkä luokassa 1 (RQ2)?. EU:n uuden lainsäädännön Medical Device Regulation:in (MDR:n) ja sen luokittelusäännön mukaisesti yksikään ohjelmistopohjainen lääkinnällinen laite ei voi saada alempaa luokitusta kuin 2a, joten käytännössä kaikki edellisen lainsäädännön (Medical Device Directive) aikana merkityt luokan 1 lääkinnälliset laitteet nousevat riskiluokassa ylöspäin vähintään luokkaan 2a. Tämä tuo mm. vaatimuksen ilmoitetun laitoksen auditoinnista kaikille ohjelmistopohjaisille lääkinnällisille laitteille ja näin ol-

len kasvattaa myös regulaation määrää.

Tutkielman pääpaino oli kuitenkin tutkia, voidaanko ohjelmistopohjaisten lääkinnällisten laitteiden jatkuva integraatio ja tuotantoonvienti saavuttaa, ja miten (RQ3)? Tätä aihealuetta rajattiin niin, että tutkielma painottuu IEC 62304-standardin mukaisen ohjelmistokehityksen vaiheisiin. Emme siis ottaneet tutkielmassa kantaa lääkinnällisiin laitteisiin liittyviin muihin aktiviteetteihin, jotka eivät olleet IEC 62304 piirissä, ja nämä aktiviteetit kattavat mm. käytettävyystestauksen, validoinnin ja riskienarvioinnin.

IEC 62304-standardi kuvaa ohjelmistokehityksen eri vaiheet ja niiden vaatimukset. Tässä tutkielmassa esitetty osuus koskeekin nimenomaan kyseisen standardin esittelemän kehityksen vaiheita ja automaatioita, johon itse ohjelmistomoduulien ja -yksikköjen kehitys, integraatio ja testaus kuuluvat.

Pystyimme osoittamaan, että tällä rajauksella voidaan saavuttaa lähes sama ohjelmistokehityksen velositeetti reguloidussa ohjelmistokehityksessä kuin reguloimattomassa ohjelmistokehityksessä. Suurimman eron näiden välille tuo vaatimus automaation kehittämiselle sekä dokumentoinnille, jotka eivät ole pakollisia vaatimuksia ei-reguloidussa ohjelmistokehityksessä. Automaation, kuten automaatiotestien, rakentaminen ja dokumentointi kuitenkin hyödyttää myös ei-reguloitua ohjelmistokehitystä parantaen sen laatua.

Pystyimme osoittamaan, että jatkuvasta integraatiosta ja tuotantoonviennistä on hyötyä ohjelmistopohjaisten lääkinnällisten laitteiden kehityksessä, ja niiden automaation avulla jatkuva kehittäminen on ylipäätään mahdollista. Tämä mahdollistaa DevOps-toimintatavan mukaisen iteratiivisen kehityksen ja sen ketteryyden.

Todellisuudessa ensimmäisestä ideasta tuotantoon asennukseen ohjelmistopohjaisen lääkinnällisen laitteen kehitys on hitaampaa kuin ei-reguloidun ohjelmiston kehitys, johtuen tämän tutkielman ulkopuolella olevista aktiviteeteista. **Kun kuitenkin tarkastelu keskitetään ohjelmiston toteutus- ja verifiointivaiheeseen, on mahdollista saavuttaa sama lähes sama velositeetti kuin reguloimattomassa ohjelmistokehityksessä.**

Tutkimuksen jatkaminen: Tässä työssä esitettyä prosessia ei ole sertifioitu tai auditoitu ilmoitetun laitoksen toimesta, minkä vuoksi yksi tärkeistä jatkotukimuksen kohteista olisi tuottaa todellisen maailman tutkimus, jossa vertailtaisiin useita toimijoita, jotka tuottavat ohjelmistopohjaisia lääkinnällisiä laitteita EU:n Medical Device Regu-

lationin mukaisesti. Toistaiseksi tällaisia tutkimuksia ei ole tehty ja tutkimusten toteutus tulee olemaan helpompaa arviolta vuonna 2021, kun ohjelmistopohjaisten 2a-luokan lääkinnällisten laitteiden määrä tulee kasvamaan luokittujen laitosten lisääntymisen ja MDR:n siirtymäajan loppumisen myötä.

Tutkimuksella, jossa vertailtaisiin useita ohjelmistopohjaisten lääkinnällisten laitteiden valmistajia ja teollisen mittakaavan omaavien ei-reguloitujen ohjelmistojen valmistajia, voitaisiin osoittaa oikean maailman todisteiden valossa ohjelmistokehityksen velositeetin erot. Vertailuun ei ole mielekästä ottaa ei-ammattimaisesti toimivia ohjelmistovalmistajia, sillä teknisesti ohjelmistokehitystä voi tehdä lähes ilman mitään prosessia, esimerkiksi suoraan tuotantoon. Tällainen malli on vääjäämättä ketterämpi kuin mikään muu ohjelmistokehitysprosessi, mikäli mittarina toimii tuotantoonvientonopeus.

Mielekästä olisi myös tutkia lääkinnällisten laitteiden valmistamisen muita vaiheita, joihin kuuluu myös esimerkiksi validointi ja käytettävyystestaus. On todennäköistä, että jatkuva integraatio ja tuotantoonvienti voisi nopeuttaa myös näitä vaiheita lääkinnällisten laitteiden kehityksessä, vaikka täyttä automaatiota tuskin voidaankaan saavuttaa.

Kirjallisuus

Wang, T., Wei, T., Gu, G. ja Zou, W. (2010). ”TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection”. Teoksessa: *2010 IEEE Symposium on Security and Privacy*, s. 497–512. DOI: 10.1109/SP.2010.37.

Abrahamsson, P., Ebert, C. ja Oza, N. (2012). ”Lean Software Development”. *IEEE Software* 29.05, s. 22–25. ISSN: 1937-4194. DOI: 10.1109/MS.2012.116.

Agile Adoption Rate Survey Results: February 2008 (2008). URL: http://www.ambysoft.com/surveys/agileFebruary2008.html#Downloads (viitattu 08. 09. 2019).

Ahmad, M. O., Markkula, J. ja Oivo, M. (2013). ”Kanban in software development: A systematic literature review”. Teoksessa: *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, s. 9–16. DOI: 10.1109/SEAA.2013.28.

Balaji, S. ja Murugaiyan, M. S. (2012). ”Waterfall vs. V-Model vs. Agile: A comparative study on SDLC”. *International Journal of Information Technology and Business Management* 2.1, s. 26–30.

Beck, K. M., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S. J., Schwaber, K., Sutherland, J. ja Thomas, D. (2013). ”Manifesto for Agile Software Development”.

Bitbucket Pipelines (2019). https://confluence.atlassian.com/bitbucket/build-test-and-deploy-with-pipelines-792496469.html. (Accessed on 10/27/2019).

Cook, S. P., Angermayer, J., Lacher, A., Buttner, A., Crouse, K. ja Lester, E. (2015). ”Dependability of Software of Unknown Pedigree: Case studies on unmanned aircraft systems”. Teoksessa: *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, s. 1–20. DOI: 10.1109/DASC.2015.7311440.

Day, S. ja Zweig, M. (2018). *2018 Year End Funding Report: Is digital health in a bubble? — Rock Health — We're powering the future of healthcare. Rock Health is a seed and early-stage venture fund that supports startups building the next generation of technologies transforming healthcare.* https://rockhealth.com/reports/2018-year-end-funding-report-is-digital-health-in-a-bubble/. (Accessed on 09/14/2019).

Decan, A., Mens, T. ja Claes, M. (2017). ”An empirical comparison of dependency issues in OSS packaging ecosystems”. Teoksessa: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, s. 2–12. DOI: 10.1109/SANER.2017.7884604.

Filion, L., Daviot, N., Le Bel, J. ja Gagnon, M. (2017). ”Using Atlassian tools for efficient requirements management: An industrial case study”. Teoksessa: *2017 Annual IEEE International Systems Conference (SysCon)*, s. 1–6. DOI: 10.1109/SYSCON.2017.7934769.

Food, U. ja Administration, D. (2018). *Medical Device Classification Procedures*. [https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRsearch.cfm?CFRPart=860, 10.7.2019].

Healthcare Software Investment Trends - Digital Health Matters (2017). http://www.norailuri.com/healthcare-software-investment-trends/. (Accessed on 02/21/2020).

IEC 62304 (2015). ”Medical device software - Software life cycle processes”.

IEC 62366 (2015). ”Medical diveces, Application of usability engineering to medical devices”.

ISO 13485 (2016). ”ISO 13485 Medical devices - Quality management systems - Requirements for regulatory purposes”.

ISO 14971 (2009). ”Medical diveces, application of risk management to medical devices”.

Lwakatare, L. E., Karvonen, T., Sauvola, T., Kuvaja, P., Holmström, H., Bosch, O. J. ja Oivo, M. (2016). ”Towards DevOps in the Embedded Systems Domain: Why is It so Hard?” *HICSS 49th Hawaii International Conference on System Sciences*.

Laukkarinen, T., Kuusinen, K. ja Mikkonen, T. (2017). ”DevOps in Regulated Software Development: Case Medical Devices”. *ICSE-NIER ’17 Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*, s. 15–18.

Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V., Itkonen, J., Mäntylä, M. V. ja Männistö, T. (2015). ”The highways and country roads to continuous deployment”. *IEEE Software* 32.2, s. 64–72. ISSN: 1937-4194. DOI: 10.1109/MS.2015.50.

McHugh, M., Cawley, O., McCaffery, F., Richardson, I. ja Wang, X. (2013). ”An Agile V-model for Medical Device Software Development to Overcome the Challenges with Plan-driven Software Development Lifecycles”. Teoksessa: *Proceedings of the*

5th International Workshop on Software Engineering in Health Care. SEHC '13. San Francisco, California: IEEE Press, s. 12–19. ISBN: 978-1-4673-6282-5.

MDR Guidance, European Commission (2019). *Guidance on Qualification and Classification of Software in Regulation (EU) 2017/745 – MDR and Regulation (EU) 2017/746 – IVDR*. https://ec.europa.eu/docsroom/documents/37581?locale=en. (Accessed on 11/17/2019).

Medical Device Directive, European Union (1993). *Article IX of the Council Directive 93/42/EEC*. [https://eur-lex.europa.eu/legal-content/FI/TXT/PDF/?uri=CELEX:52017AG0002(01)&from=FI, 10.7.2019].

Medical Device Regulation, European Union (2017). *Regulation (EU) 2017/745 of the European Parliament and of the Council of 5 April 2017 on medical devices, amending Directive 2001/83/EC*. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32017R0745. (Accessed on 09/21/2019).

Meyer, M. (2014). "Continuous Integration and Its Tools". *IEEE Software* 31.3, s. 14–16. DOI: 10.1109/MS.2014.58.

Smeds, J., Nybom, K. ja Porres, I. (2015). "DevOps: A Definition and Perceived Adoption Impediments". Teoksessa: *Agile Processes in Software Engineering and Extreme Programming*. Toim. C. Lassenius, T. Dingsøyr ja M. Paasivaara. Cham: Springer International Publishing, s. 166–177. ISBN: 978-3-319-18612-2.

Tsai, W. T., Xiaoying Bai, Paul, R., Weiguang Shao ja Agarwal, V. (2001). "End-to-end integration testing design". Teoksessa: *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*, s. 166–171. DOI: 10.1109/CMPSAC.2001.960613.